Appendixes

Contents

Appendix A.
DOS Version 2.00 Enhancements

The information in this appendix is divided into two categories—those topics that apply to all users, and those topics that apply to system programmers or application developers. In each case, a brief description of the feature or change is offered, and you are referred to another section of the book for further details.

For All Users

DOS Version 2.00 incorporates the following new and changed features:

- *Special characters.* The characters <, >, |, and \ now have special meanings to DOS, and can no longer be used in filenames. If you have files whose names contain any of these characters, they should be renamed (using your *old* version of DOS) before attempting to use them with DOS Version 2.00.

- *Configuration file.* You can create a file of special commands that DOS will read each time it starts up. The commands allow you to specify the number of disk buffers DOS should use, the names of device drivers, and additional information concerning DOS operation. Please refer to "Configuring Your System" in Chapter 5 (Section 1) for additional information.

- *Support for one or more fixed disk devices.* The disk can be divided into multiple partitions, each usable by a different operating system. You can start (boot) your operating system from the fixed disk, and utility programs included to perform disk initialization, backup, and restore functions. If you have a fixed disk, please read the "Preparing Your Fixed Disk" information in Chapter 4 for setup instructions and the BACKUP and RESTORE commands in Chapter 6 for their respective functions.

- *Support for increased diskette capacity.* Beginning with DOS Version 2.00, DOS formats diskettes at 9 sectors per track, which increases capacity from 163840 to 184320 characters of information for single-sided diskettes, and from 327680 to 368640 characters for dual-sided diskettes. The smaller capacity diskettes created by DOS Version 1.00 or DOS Version 1.10 (8 sectors per track) are also usable with DOS Version 2.00. You do not need to reformat them. Please see the FORMAT and DISKCOPY commands in Chapter 6 for more information.

- *Multiple disk buffers.* A *disk buffer* is an area of user memory that DOS reserves at startup and is used for performing disk and diskette operations. DOS normally allocates two disk buffers at start-up time. Some users, however, will find that certain applications, such as data base applications, may run faster if DOS has more buffers available to it. DOS Version 2.00 allows you to specify the number of buffers that DOS should reserve at start-up time. Please refer to "Configuring Your System" in Chapter 9 for instructions on specifying additional buffers.

- *Tree-Structured Directories.* This new feature allows you to place related groups of files in their own directories—all on the same disk. The individual directories are isolated from each other, giving the appearance of separate disks. Therefore, a search for a file in a given directory will not "see" files in other directories on the same disk.

  Each directory, beginning with the normal system directory (called the *root directory*) may contain special entries naming other directories on the same disk. These other directories, in turn, may contain entries for even more directories, and so on. When viewed in a logical order beginning with the root directory, the directory structure appears much like a diagram of a family tree—thus the term *tree-structured directories.*

  You may add or remove directories, copy files from one directory to another, instruct DOS to look in a specific directory to locate a file, etc. For complete details, please refer to Chapter 5 "Using Tree-Structured Directories".

- *Disk Volume Labels.* This feature allows you to specify a unique volume label (up to 11 characters) at the time you format a disk. The volume label is placed in the root directory, and is included in the displays produced by the DIR, CHKDSK, and TREE commands. Please refer to the FORMAT command for further information.

- *Extended DOS screen and keyboard control.* This feature allows you to issue special character sequences from within your program that DOS will use for screen cursor positioning and color, and further allows you to assign the meaning of any key on the keyboard. For example, you may assign the character string "DIR A:" to the F10 key so that simply pressing the F10 key has the same result as entering the **DIR A:** command. Please refer to "Using Extended Screen and Keyboard Control" in Chapter 13, and "Configuring Your System" in Chapter 9 for more detailed information.

- *Redirection of Standard Input and Output.* This feature applies to all DOS programs that read from the keyboard or write to the screen (the standard input and output devices). By using the special characters < (for input) and > (for output), you can cause a program to receive its input from a source other than the keyboard, or to direct its output to a destination other than the screen. For example, the command:

```
DIR A:>DIRLIST
```

Causes the directory listing from drive A to be placed in a file named DIRLIST on the default drive. Device names can also be used. For example, the command:

```
DIR A:>PRN
```

Causes the directory listing to appear on the printer instead of the screen. Please refer to "Redirection of Standard Input and Output" in Chapter 10 for further information.

- *Piping of standard input and output.* This feature allows the standard output of one program to be used as the standard input to another. DOS acts as a "pipeline" to direct the output of the first program to the input of the second—thus, the *term* "piping." For further information and an example of its use, please refer to "Piping of Standard Input and Output" in Chapter 10.

New Commands

The following new commands have been added to DOS Version 2.00. Please consult the command descriptions in Chapter 6 and Chapter 10 for further details and examples of their use.

ASSIGN

Allows you to reassign drive letters so that a request for a given drive can be routed to a different drive.

BACKUP

Backs up one or more files from a fixed disk to diskettes.

BREAK

Allows you to specify when DOS should check for a Ctrl-Break being entered at the keyboard. Normally, DOS only performs this check during screen, keyboard, printer, or auxiliary device operation. With this command, you can instruct DOS to check for Ctrl-Break whenever a program requests DOS to perform *any* function (such as disk operations). In this way, it is possible to "break out" of a program that performs few or no screen or keyboard operations (such as a compiler).

CLS

Clears the screen when used from a batch file or the keyboard.

CTTY

Allows you to define a different primary console, so that a remote terminal device can be used in place of the standard screen and keyboard. This command also reverses this assignment, to restore the keyboard and screen as the standard input and output devices.

ECHO, IF, FOR, SHIFT, GOTO

New subcommands provided to extend the flexibility of batch processing.

FDISK

Initializes and configures a fixed disk.

> **Note:** This command must be used before you use your fixed disk for the first time. Please refer to Chapter 4 "Preparing Your Fixed Disk".

GRAPHICS

Allows the Shift-PrtSc keys (display screen contents on printer) to print the image of a graphics display screen.

MKDIR, RMDIR and CHDIR

Create, remove, and inform DOS to use a directory other than the system directory.

PATH

Allows you to specify one or more paths of directory names that DOS will search if the command you have issued was not found in the current directory. They allow conditional execution of commands within a batch file by causing DOS to check for specified conditions.

PRINT

Prints a queue (list) of files on the system printer while you are using the system for other work.

PROMPT

Allows you to change the system prompt to a desired string.

RECOVER

Recovers a specific file that cannot be copied or otherwise used because of a defective spot on the disk that prevents the file from being read. This command also recovers multiple files when the directory has been damaged.

RESTORE

Restores one or more files from diskettes to a fixed disk.

SET

Allows you to enter keywords and parameters into a DOS "environment" that is accessible by commands and applications.

TREE

Displays the entire directory structure of the specified disk.

VER

Displays the DOS version number on the screen.

VERIFY

Instructs DOS to perform a verify operation (or to stop performing the verify) each time data is written to disk, until a new verify command is issued to turn the verify feature off. The verify operation increases assurance that the data was properly recorded on disk (that is, it can be read without error).

VOL

Displays the volume label of the disk in the specified drive.

Enhanced Commands

The following commands, that existed in DOS Version 1.10, have been enhanced for DOS Version 2.00. For more detailed information, please consult the individual command descriptions in Chapter 6 and Chapter 10.

CHKDSK

Supports the fixed disk, the new and old diskette formats, and analyzes all directories on the volume. It also enables you to create files containing all sectors that were found to be allocated but were not associated with a file, so that you can recover "lost" data. An important feature is that, unlike Version 1.10 CHKDSK, it will take *no* corrective action on the disk being analyzed unless instructed to do so.

COMP

The file compare utility now allows multiple files to be compared. For example, you can compare all of the files on one disk with their counterparts on another disk. Also, COMP no longer prompts you to insert diskettes before comparing.

DEBUG

Now contains a command allowing you to enter assembly language statements that are assembled directly into memory.

DIR

Now displays the volume identification of the specified disk and clearly identifies entries that contain the names of other directories. It also displays the amount of available space left on the disk.

DISKCOPY and DISKCOMP

Support the new 9-sector-per-track diskette format.

EDLIN

The line editor contains several new subcommands for more flexible management of source data. They include commands to copy and move lines, and to merge the contents of another file. The Replace and Search commands have been changed to begin their search at the current line plus one.

ERASE

Now requires you to press the Enter key after entering the Y/N response to the

Are you sure

message that appears when you instruct DOS to erase all of the files on a volume. This is intended to prevent accidental erasure of all files from the larger capacity devices supported by DOS Version 2.00.

FORMAT

Now formats diskettes at 9 sectors per track (in DOS 1.00 and 1.10 diskettes were formatted at 8 sectors per track), allowing each new diskette to hold more data. It also allows you to specify a volume identification that is recorded in the disk's directory. Support for initializing a fixed disk is also included.

LPT2:, LPT3:, and COM2:

Now recognized as valid device names by DOS, and can be used in place of filenames.

For Programmers

- DOS Version 2.00 includes the ability to install your own device drivers for character or block-oriented devices. Please refer to "Installable Device Drivers" in Chapter 14 for more details.

- Three changes were made to internal functions that cause different results from those obtained on DOS Version 1.10:

  1. The function call (hex 1B) that previously returned a pointer to the file allocation table now returns a pointer to only the table's identification byte, for purposes of determing the disk type. All applications that use call hex 1B to obtain the file allocation table should be changed to use interrupt hex 25 to read the file allocation table directly from the disk. The file allocation table always begins at logical sector 1, and its size can be determined from the information returned by call hex 36. We recommend that you avoid using calls hex 1B and hex 1c.

  2. The mapping of logical sectors on dual sided diskettes has been rearranged to facilitate program loading and to improve system performance.

     This change allows DEBUG to load an entire file with a single L command.

     Applications that use interrupts hex 25 and hex 26 on multi-sided disks or diskettes may require modification to operate properly on DOS Version 2.00. Please see the description of Int 25 in Appendix D.

3. Additional bits have been defined in the file attribute byte of the DOS disk directory. Programs that depended upon the file attribute byte being equal to zero if a file was not a hidden or system file may not work correctly. (See Appendix C for the definition of the new attribute bits.)

- A new set of function calls has been made available to provide a wide variety of services. We suggest that systems programmers and application developers review all of Appendix D for details on these new functions.

Notes:

Appendix B.
DOS Technical Information

Appendixes B—K are intended to supply technically oriented users with information about the structure, facilities, and program interfaces of DOS. It is assumed that the reader is familiar with the 8088 architecture, interrupt mechanism, and instruction set.

DOS Structure

DOS consists of the following four components:

1. The boot record resides on track 0, sector 1, side 0 of every disk formatted by the FORMAT command. It is put on all disks in order to produce an error message if you try to start up the system with a non-DOS diskette in drive A. For fixed disks, it resides on the first sector (sector 1, head 0) of the first cylinder of the DOS partition.

2. The Read-Only Memory (ROM) BIOS interface module (file IBMBIO.COM) provides a low-level interface to the ROM BIOS device routines.

3. The DOS program itself (file IBMDOS.COM) provides a high-level interface for user programs. It consists of file management routines, data blocking/deblocking for the disk routines, and a variety of built-in functions easily accessible by user programs. (Refer to Appendix D.)

   When these function routines are invoked by a user program, they accept high-level information via register and control block contents, then (for device operations) translate the requirement into one or more calls to IBMBIO to complete the request.

4. The command processor, COMMAND.COM.

DOS Initialization

When the system is started (either System Reset or power ON with the DOS diskette in drive A), the boot record is read into memory and given control. It checks the directory to assure that the first two files listed are IBMBIO.COM and IBMDOS.COM, in that order. (An error message is issued if not.) These two files are then read into memory. (IBMBIO.COM must be the first file in the directory, and its sectors must be contiguous.)

The initialization code in IBMBIO.COM determines equipment status, resets the disk system, initializes the attached devices, causes device drivers to be loaded, and sets the low-numbered interrupt vectors. It then relocates IBMDOS.COM downward and calls the first byte of DOS.

As in IBMBIO.COM, offset 0 in DOS contains a jump to its initialization code, which will later be overlaid by a data area and the command processor. DOS initializes its internal working tables, initializes interrupt vectors for interrupts hex 20 through hex 27 and builds a Program Segment Prefix (see Appendix E) for COMMAND.COM at the lowest available segment, then returns to IBMBIO.COM.

The last remaining task of initialization is for IBMBIO.COM to load COMMAND.COM at the location set up by DOS initialization. IBMBIO.COM then passes control to the first byte of COMMAND.

The Command Processor

The command processor supplied with DOS (file COMMAND.COM) consists of four distinctly separate parts:

- A resident portion resides in memory immediately following IBMDOS.COM and its data area. This portion contains routines to process interrupt types hex 22 (terminate address), hex 23 (CTRL-BREAK handler), and hex 24 (critical error handling), as well as a routine to reload the transient portion if needed. (When a program terminates, a checksum methodology determines if the program had caused the transient portion to be overlaid. If so, it is reloaded.) Note that all standard DOS error handling is done within this portion of COMMAND. This includes displaying error messages and interpreting the reply of Abort, Retry, or Ignore. (See message **Disk error reading drive *x*** in Chapter 8.)

- An initialization portion follows the resident portion and is given control during startup. This section contains the AUTOEXEC file processor setup routine. The initialization portion determines the segment address at which programs can be loaded. It is overlaid by the first program COMMAND loads because it's no longer needed.

- A transient portion is loaded at the high end of memory. This is (portion 3) the command processor itself, containing all of the internal command processors, the batch file processor, and (portion 4) a routine to load and execute external commands (files with filename extensions of .COM or .EXE). this "loader" is at the highest end of memory, and is invoked by the EXEC function call to load programs.

  Portion 3 of COMMAND produces the system prompt (such as A>), reads the command from the keyboard (or batch file) and causes it to be executed. For external commands, it builds a command line and issues an EXEC function call to load and transfer control to the program.

Appendix E contains detailed information describing the conditions in effect when a program is given control by EXEC.

Available DOS Functions

DOS provides a significant number of functions to user programs, all available through issuance of a set of interrupt codes. There are routines for keyboard input (with and without echo and Ctrl-Break detection), console and printer output, constructing file control blocks, memory management, date and time functions, and a variety of disk, directory, and file handling functions. See "DOS Interrupts and Function Calls" in Appendix D for detailed information.

File Management Notes

Through the INT 21 (function call) mechanism, DOS provides methods to create, read, write, rename, and erase files. Files are not necessarily written sequentially on disk—space is allocated as it is needed, and the first location available on the disk is allocated as the next location for a file being written. Therefore, if considerable file creation and erasure activity has taken place, newly created files will probably *not* be written in sequential sectors.

However, due to the mapping (chaining) of file space via the File Allocation Table, and the function calls available, any file can be used in either a sequential or random manner.

There are two sets of function calls that support file management. The new, extended set of calls is the preferred method (functions 39 through 57). Through these calls, sequential and random file accesses are simpler than using the traditional (FCB oriented) set of calls. The FCB calls continue to function as in the past: By using the current block and current record fields of the FCB, and the sequential disk read or write functions, you can make the file appear sequential—DOS will do the calculations necessary to locate the proper sectors on the disk. On the other hand, by using the random record field, and random disk functions, you can cause any record in the file to be accessed *directly*—again, DOS will locate the corrected sectors on the disk for you.

Space is allocated in increments called *clusters*. For single sided diskettes, this unit of allocation is one sector; for dual sided diskettes, each cluster is two consecutive sectors in length. The cluster size of a fixed disk is determined at FORMAT time, and is based on the size of the DOS partition.

The Disk Transfer Area (DTA)

The Disk Transfer Area (also commonly called *buffer*) is the memory area DOS will use to contain the data for all file reads and writes that are performed with the traditional (FCB) set of function calls. This area can be at any location within memory, and should be set by your program. (See function call hex 1A.)

Only one DTA can be in effect at a time, so it is the program's responsibility to inform DOS what memory location to use *before* using any disk read or write functions. Once set, DOS continues to use that area for all disk operations until another function call hex 1A is issued to define a new DTA. When a program is given control by COMMAND, a default DTA has already been established at hex 80 into the program's Program Segment Prefix, large enough to hold 128 bytes.

When using the extended file management function calls, you specify a buffer address when you issue the read or write call. There is no need to set a DTA address.

Error Trapping

DOS provides a method by which a program can receive control whenever a disk or device read/write error occurs, or when a bad memory image of the file allocation table is detected. When these events occur, DOS executes an INT hex 24 to pass control to the error handler. The default error handler resides in COMMAND.COM, but any program can establish its own by setting the INT hex 24 vector to point to the new error handler. DOS provides error information via the registers and provides Abort, Retry, or Ignore support via return codes. (Refer to Appendix D. "DOS Interrupts and Function Calls".)

Notes:

Appendix C. DOS Disk Allocation

All disks and diskettes formatted by DOS are created with a sector size of 512 bytes. The DOS area (entire diskette for diskettes, DOS partition for fixed disks) is formatted as follows:

Boot record – variable size
First copy of file allocation table – variable size
Second copy of file allocation table – variable size
Root directory – variable size
Data area

Allocation of space for a file (in the data area) is done only when needed (it is not pre-allocated). The space is allocated one cluster (unit of allocation) at a time. A cluster is always one or more consecutive sectors, and all of the clusters for a file are "chained" together in the File Allocation Table.

The clusters are arranged on disk to minimize head movement for multi-sided media. All of the space on a track (or cylinder) is allocated before moving on to the next track. This is accomplished by using the sequential sectors on the lowest-numbered head, then all the sectors on the next head, and so on until all sectors on all heads of the track are used. Then, the next sector to be used will be sector 1 on head 0 of the next track.

For fixed disk, the size of the file allocation table and directory are determined when FORMAT initializes it, and are based on the size of the DOS partition.

For diskettes, the following table can be used:

# Sides	Sectors/ Track	FAT size Sectors	Dir Sectors	Dir Entries	Sectors/ Cluster
1	8	1	4	64	1
2	8	1	7	112	2
1	9	2	4	64	1
2	9	2	7	112	2

Files in the data area are not necessarily written sequentially on the disk. The data area space is allocated one cluster at a time, skipping over clusters already allocated. The first free cluster found will be the next cluster allocated, regardless of its physical location on the disk. This permits the most efficient utilization of disk space because clusters made available by erasing files can be allocated for new files. (Refer to the description of the "DOS File Allocation Table".)

DOS Disk Directory

FORMAT initially builds the root directory for all disks. Its location (logical sector number) and the maximum number of entries are available through the device driver interfaces.

Since directories other than the root directory are actually files, there is no limit to the number of entries they may contain. Sub-directories can be read as data files, using an extended FCB with the appropriate attribute byte.

All directory entries are 32 bytes in length, and are in the following format (byte offsets are in decimal):

0-7 Filename. The first byte of this field indicates its status.

hex 00 Never been used. This is used to limit the length of directory searches, for performance reasons.

hex E5 Was used, but the file has been erased.

hex 2E The entry is for a directory. If the second byte is also hex 2E, then the cluster field contains the cluster number of this directory's parent directory (hex 0000 if the parent directory is the root directory).

Any other character is the first character of a filename.

8-10 Filename extension.

11 File attribute. The attribute byte is mapped as follows (values are in hexadecimal):

01 File is marked read-only. An attempt to open the file for output using function call hex 3D results in an error code being returned. This value can be used along with other values below.

02 Hidden file. The file is excluded from normal directory searches.

04 System file. The file is excluded from normal directory searches.

08 The entry contains the volume label in the first 11 bytes. The entry contains no other usable information, and may exist only in the root directory.

10 The entry defines a sub-directory, and is excluded from normal directory searches.

20 Archive bit. The bit is set on whenever the file has been written to and closed. It is used by backup/restore functions of the FDISK utility for determining whether or not the file was changed since it was last backed up. This bit can be used along with other attribute bits.

Note: The system files (IBMBIO.COM and IBMDOS.COM) are marked as read only, hidden, and system files. Files can be marked hidden when they are created. Also, the read-only, hidden, system, and archive attributes may be changed through the CHMOD function call.

12-21 Reserved.

22-23 Time the file was created or last updated. The time is mapped in the bits as follows:

```
<    hh    ><    mm    ><    xx    >
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
```

where:

hh is the binary number of hours (0-23)
mm is the binary number of minutes (0-59)
xx is the binary number of two-second increments

24-25 Date the file was created or last updated. The mm/dd/yy are mapped in the bits as follows:

```
<          25          ><          24          >
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
y  y  y  y  y  y  y m m m m d d d d d
```

where:

mm is 1-12
dd is 1-31
yy is 0-119 (1980-2099)

26-27 Starting cluster; the relative cluster number of the first cluster in the file.

Note that the first cluster for data space on all fixed disks and diskettes is always cluster 002.

The cluster number is stored with the least significant byte first.

Note: System programmers, see "DOS File Allocation Table" for details about converting cluster numbers to logical sector numbers.

28-31 File size in bytes. The first word contains the low-order part of the size. Both words are stored with the least significant byte first.

DOS File Allocation Table

This information is presented for the benefit of system programmers who wish to develop device drivers. It explains how DOS uses the File Allocation Table to convert the clusters of a file to logical sector numbers. The driver is then responsible for locating the logical sector on disk. We wish to emphasize that this information should not be used for any other purpose. We recommend that system utilities use the DOS file management function calls rather than interpreting the FAT.

The File Allocation Table (FAT) is used by DOS to allocate disk space for a file, one cluster at a time.

The FAT consists of a 12-bit entry (1.5 bytes) for each cluster on the disk.

Note that the first two FAT entries map a portion of the directory; these FAT entries contain indicators of the size and format of the disk.

The second and third bytes always contain hex FFFF. The first byte is used as follows:

Hex Value	Meaning
FF	Dual sided, 8 sector-per-track diskette.
FE	Single sided, 8 sector-per-track diskette.
FD	Dual sided, 9 sector-per-track diskette.
FC	Single sided, 9 sector-per-track diskette.
F8	Fixed disk

The third FAT entry begins the mapping of the data area (cluster 002).

Each entry contains three hexadecimal characters, either:

000	if the cluster is unused and available, or
FF8-FFF	to indicate the last cluster of a file, or
XXX	any other hexadecimal characters that are the cluster number of the *next cluster* in the file. The cluster number of the first cluster in the file is kept in the file's directory entry.

Note: The values FF0-FF7 are used to indicate reserved clusters (FF7 indicates a bad cluster if it is not part of an allocation chain), and FF8-FFF are used as end-of-file marks.

The File Allocation Table always begins on logical sector 1 (second actual sector on a diskette or in a fixed disk partition), following the boot record. If larger than 1 sector, the sectors are continguous. Two copies of the FAT are written, one following the other, for integrity. The FAT is read into one of the DOS buffers whenever needed (open, allocate more space, etc.), and that buffer is given a high priority to keep it in memory as long as possible, for performance reasons.

How to Use the File Allocation Table

Obtain the *starting cluster* of the file from the directory entry.

Now, to locate each subsequent cluster of the file:

1. Multiply the cluster number just used by 1.5 (each FAT entry is 1.5 bytes long).

2. The whole part of the product is an offset into the FAT, pointing to the entry that maps the cluster just used. That entry contains the cluster number of the next cluster of the file.

3. Use a MOV instruction to move the word at the calculated FAT offset into a register.

4. If the last cluster used was an even number, keep the low-order 12 bits of the register; otherwise, keep the high-order 12 bits.

5. If the resultant 12 bits are hex FF8-FFF, there are no more clusters in the file. Otherwise, the 12 bits contain the cluster number of the next cluster in the file.

To convert the cluster to a logical sector number (relative sector, such as that used by Int 25 and 26 and by DEBUG):

1. Subtract 2 from the cluster number.

2. Multiply the result by the number of sectors per cluster.

3. Add the logical sector number of the beginning of the data area.

Notes:

Interrupts

Note: We recommend that a program wishing to examine or set the contents of any interrupt vector use the DOS function calls (hex 35 and hex 25) provided for those purposes, and avoid referencing the interrupt vector locations directly.

DOS reserves interrupt types hex 20 to hex 3F for its use. This means absolute memory locations hex 80 to hex FF are reserved by DOS. The defined interrupts are as follows with all values in hexadecimal.

20 Program terminate. Issuing Interrupt hex 20 is the traditional way to exit from a program. This vector transfers to the logic in DOS for restoration of the terminate, Ctrl-Break, and critical error exit addreses to the values they had on entry to the program. All file buffers are flushed. All files changed in length should be closed (see function call hex 10 and hex 3E) prior to issuing this interrupt. If the changed file is not closed, its length, date, and time are not recorded correctly in the directory.

In order for a program to pass a completion (or error) code when terminating, it must use either function call hex 4C (exit) or hex 31 (terminate and stay resident). These two new methods are preferred over using interrupt hex 20, and the codes returned by them can be interrogated in batch processing (see ERRORLEVEL subcommand of batch processing).

Important: Every program must ensure that the CS register contains the segment address of its Program Segment Prefix control block prior to issuing interrupt hex 20.

21 Function request. Refer to "Function Calls" in this appendix.

22 Terminate address. The address represented by this interrupt is the address to which control transfers when the program terminates. This address is copied into the program's Program Segment Prefix at the time the segment is created. If a program wishes to execute a second program it must set the terminate address prior to issuing the EXEC function call to execute the new program. Otherwise, when the second program executes, its termination would cause transfer to its host's termination address. This address, as well as the Ctrl-Break address below, may be set via DOS function call hex 25.

23 Ctrl-Break exit address. If the user enters Ctrl-Break during screen, printer, or asynchronous communications adapter operations, an interrupt type hex 23 is executed. (If BREAK is on, the interrupt hex 23 is issued on *any* function call.) If the Ctrl-Break routine saves all registers, it may end with a return-from-interrupt instruction (IRET) to continue program execution. If the program returns with a long return, the carry flag is used to determine whether the program will be aborted or not; if the carry flag is set, it will be aborted, otherwise execution will continue (as with a return by IRET). If the Ctrl-Break interrupts functions 9 or 10, buffered I/O, then C, carriage-return, and linefeed are output. If execution is then continued with an IRET, I/O continues from the start of the line. When the interrupt occurs, all registers are set to the value they had when the original function call to DOS was made. There are no restrictions on what the Ctrl-Break handler is allowed to do, including DOS function calls, as long as the registers are unchanged if IRET is used.

If the program creates a new segment and loads in a second program which itself changes the Ctrl-Break address, the termination of the second program and return to the first causes the Ctrl-Break address to be restored to the value it had before execution of the second program. (It is restored from the second program's Program Segment Prefix.)

24 Critical error handler vector. When a critical error occurs within DOS, control is transferred with an interrupt 24H. On entry to the error handler, AH will have its bit 7=0 (high-order bit) if the error was a disk error (probably the most common occurrence), bit 7=1 if not.

BP:SI contains the address of a Device Header Control Block from which additional information can be retrieved (see below).

The registers will be set up for a retry operation, and an error code will be in the lower half of the DI register with the upper half undefined. These are the error codes:

Error Code	Description
0	Attempt to write on write-protected diskette
1	Unknown unit
2	Drive not ready
3	Unknown command
4	Data error (CRC)
5	Bad request structure length
6	Seek error
7	Unknown media type
8	Sector not found
9	Printer out of paper
A	Write fault
B	Read fault
C	General failure

The user stack will be in effect (the first item described below is at the top of the stack), and will contain the following from top to bottom:

IP
CS
FLAGS

DOS registers from issuing INT hex 24

AX
BX
CX
DX
SI
DI
BP
DS
ES

User registers at time of original INT hex 21 request

IP
CS
FLAGS

From the original interrupt hex 21 from the user to DOS

The registers are set such that if an IRET is executed, DOS will respond according to (AL) as follows:

(AL)=0 ignore the error.

=1 retry the operation.

=2 terminate the program through interrupt hex 23.

Disk Errors

If it is a hard error on disk (AH bit 7=0), register AL contains the failing drive number (0 = drive A, etc.); AH bits 0-2 indicate the affected disk area and whether it was a read or write operation, as follows:

Bit 0=0 if read operation,
1 if write operation

Bits 2-1 (affected disk area)

0 0 DOS area (system files)
0 1 file allocation table
1 0 directory
1 1 data area

Other Errors

If AH bit 7=1, then the error occurred on a character device, or was the result of a bad memory image of the FAT. The device header passed in BP:SI can be examined to determine which case exists. If the attribute byte high order bit indicates a block device, then the error was a bad FAT. Otherwise, the error is on a character device.

If a character device, the contents of AL are unpredictable, the error code is in DI as above.

Notes:

1. Before giving this routine control for disk errors, DOS performs five retries.

2. For disk errors, this exit is taken only for errors occurring during an interrupt hex 21 function call. It is not used for errors during an interrupt hex 25 or hex 26.

3. This routine is entered in a disabled state.

4. The SS, SP, DS, ES, BX, CX, and DX registers must be preserved.

5. This interrupt handler should refrain from using DOS function calls. If necessary, it may use calls 1 through 12. Use of any other call will destroy the DOS stack and will leave DOS in an unpredictable state.

6. The interrupt handler must not change the contents of the device header.

7. If the interrupt handler will handle errors itself rather than returning to DOS, it should restore the application program's registers from the stack, remove all but the last 3 words on the stack, then issue an IRET. This will return to the program immediately after the INT 21 that experienced the error. Note that if this is done. DOS will be in an unstable state until a function call higher than 12 is issued.

The device header pointed to by BP:SI is formatted as follows:

DWORD Pointer to next device (FFFF if last device)
WORD Attributes Bit 15 = 1 if character device, 0 if block if bit 15 is 1 Bit 0 = 1 if Current standard input Bit 1 = 1 if Current standard output Bit 2 = 1 if Current NUL device Bit 3 = 1 if Current CLOCK device Bit 14 is the IOCTL bit
WORD Pointer to Device driver strategy entry point
WORD Pointer to Device driver interrupt entry point
8-BYTE character device named field for block devices the first byte is the number of units

To tell if the error occured on a block or character device you must look at bit 15 in the attribute field (WORD at BP:SI+4).

If the name of the character device is desired, look at the eight bytes starting at BP:SI+10.

25 Absolute disk read. This transfers control directly to the DOS BIOS. Upon return, the original flags are still on the stack (put there by the INT instruction). This is necessary because return information is passed back in the current flags. Be sure to pop the stack to prevent uncontrolled growth. The request is as follows:

(AL)	Drive number (For example, 0=A or 1=B)
(CX)	Number of sectors to read
(DX)	Beginning logical sector number
(DS:BX)	Transfer address

The number of sectors specified are transferred between the given drive and the transfer address. *Logical sector numbers* are obtained by numbering each sector sequentially starting from track 0, head 0, sector 1 (logical sector 0) and continuing along the same head, then to the next head until the last sector on the last head of the track is counted. Thus, logical sector 1 is track 0, head 0, sector 2; logical sector 2 is track 0, head 0, sector 3; and so on. Numbering then continues with sector 1 on head 0 of the next track. Note that although the sectors are sequentially numbered (for example, sectors 2 and 3 on track 0 in the example above), they may not be physically adjacent on disk, due to interleaving. Note that the mapping is different from that used by DOS Version 1.10 for dual-sided diskettes.

All registers except the segment registers are destroyed by this call. If the transfer was successful the carry flag (CF) will be zero. If the transfer was not successful CF=1 and (AX) will indicate the error as follows. (AL) is the DOS error code that is the same as the error code returned in the low byte of DI when an INT hex 24 is issued, and (AH) will contain:

hex 80	Attachment failed to respond
hex 40	SEEK operation failed
hex 20	Controller failure
hex 10	Bad CRC on diskette read
hex 08	DMA overrun on operation
hex 04	Requested sector not found
hex 03	Write attempt on write-protected diskette
hex 02	Address mark not found
hex 00	Error other than types listed above

26 Absolute disk write. This vector is the counterpart of interrupt 25 above. Except for the fact that this is a write, the description above applies.

27 Terminate but stay resident. This vector is used by programs that are to remain resident when COMMAND regains control. This is the traditional method for DOS programs to remain resident upon termination.

A new DOS function call has been established that allows the terminating program to pass a completion (or error) code to DOS, that can be interpreted within batch processing (see function call hex 31). After initializing itself, the program must set DX to its last address plus one in the segment in which it is executing (the offset at which other programs can be loaded), then execute an INT 27H. DOS then considers the program as an extension of DOS, so the program is not overlaid when other programs are executed. This concept is very useful for loading programs such as user-written interrupt handlers that must remain resident.

Notes:

1. This interrupt must *not* be used by .EXE programs which are loaded into the high end of memory.

2. This interrupt restores the interrupt 22, 23, and 24 vectors in the same manner as INT 20. Therefore, it can not be used to install permanently resident Ctrl-Break or Critical Error Handler routines.

3. The maximum size of memory that can be made resident by this method is 64K. You can use call hex 31 to make a larger program resident.

28	Used internally by DOS.
29-2E	Reserved for DOS.
2F	Used internally by DOS.
30-3F	Reserved for DOS.

Function Calls

DOS provides a wide variety of function calls for character device I/O, file management, memory management, date and time functions, execution of other programs, and others. They are grouped as follows (call numbers are in hexadecimal):

0-12	Traditional character device I/O
12-24	Traditional file management
25-26	Traditional non-device functions
27-29	Traditional file management
2A-2E	Traditional non-device functions
2F-38	Extended function group
39-3B	Directory group
3C-46	Extended file management group

47 Directory group

48-4B Extended memory management group

4C-4F Extended function group

54-57 Extended function group

Functions 2F through 57 are new for DOS Version 2.00. Where similar functions exist in both this group and the group of traditional calls, we recommend using the new calls. They have been defined with simpler interfaces and provide more powerful functions than their traditional counterparts. However, if you use these new function calls your program cannot be run on DOS Versions 1.00 or 1.10.

When DOS takes control, it switches to an internal stack. User registers are preserved unless information is passed back to the requester as indicated in the specific requests. The user stack needs to be sufficient to accommodate the interrupt system. It is recommended that it be hex 80 in addition to the user needs.

Error Return Table

Many of the new function calls return the carry flag clear if the operation was successful. If an error condition was encountered, the carry flag is set, and AX contains one of the following binary error return codes:

Code	Condition
1	Invalid function number
2	File not found
3	Path not found
4	Too many open files (no handles left)
5	Access denied
6	Invalid handle
7	Memory control blocks destroyed
8	Insufficient memory
9	Invalid memory block address
10	Invalid environment
11	Invalid format
12	Invalid access code
13	Invalid data
15	Invalid drive was specified
16	Attempted to remove the current directory
17	Not same device
18	No more files

Several of the calls accept an ASCIZ string as input. This consists of an ASCII string containing an optional drive specifier, followed by a directory path, and in some cases a filename. The string is terminated by a byte of binary zeros. For example:

```
B:\LEVEL1\LEVEL2\FILE1
```

followed by a byte of zeros. Note that all calls which accept path names will accept either a forward slash or a backslash as a path separator character.

The new calls supporting files or devices use an identifier known as a "handle." When you create or open a file or device with the new calls, a 16-bit binary value is returned in AX. This is the "handle" (sometimes known as a token) that you will use in referring to the file after it's been opened.

The following handles are pre-defined by DOS and can be used by your program. You do not need to open them before using them:

0000	Standard input device. Input can be redirected.
0001	Standard output device. Output can be redirected.
0002	Standard error output device. Output cannot be redirected.
0003	Standard auxiliary device.
0004	Standard printer device.

Invoking DOS Functions

Most of the function calls require input to be passed to them in registers. After setting the proper register values, the function may be invoked in one of these ways:

1. Place the function number in AH and execute a long call to offset hex 50 in your Program Segment Prefix. Note that programs using this method will not operate correctly on DOS Versions 1.00 and 1.10.

2. Place the function number in AH and issue interrupt type hex 21.

3. There is an additional mechanism provided for pre-existing programs that were written with different calling conventions. This method should be avoided for all new programs. The function number is placed in the CL register and other registers are set according to the function specification. Then an intrasegment call is made to location 5 in the current code segment. That location contains a long call to the DOS function dispatcher. Register AX is always destroyed if this mechanism is used; otherwise, it is the same as normal function calls. This method is valid only for function calls 0-24 (hexadecimal).

The functions are as follows with all values in hexadecimal.

0 Program terminate. The terminate, Ctrl-Break, and critical error exit addresses are restored to the values they had on entry to the terminating program, from the values saved in the Program Segment Prefix. All file buffers are flushed, but any files which have been changed in length but not closed will not be recorded properly in the directory. Control transfers to the terminate address. This call performs exactly the same function as INT 20H. It is the program's responsibility to ensure that the CS register contains the segment address of its Program Segment Prefix control block prior to calling this function.

> **Note:** Calls hex 1 through hex C use the standard devices listed at the end of the "Error Return Table" in this chapter.

1 Keyboard input. Waits for a character to be read at the standard input device (unless one is ready), then echoes the character to the standard output device and returns it in AL. The character is checked for a Ctrl-Break. If Ctrl-Break is detected, an interrupt hex 23 is executed.

> **Note:** For functions 1, 6, 7, and 8, extended ASCII codes will require two function calls. (See the IBM Personal Computer BASIC manual for a description of the extended ASCII codes.) The first call returns 00 as an indicator that the next call will return an extended code.

2 Display output. The character in DL is output to the standard output device. The backspace character results in moving the cursor left one position, writing a space at this position and remaining there. If a Ctrl-Break is detected after the output, an interrupt hex 23 is executed.

3 Auxiliary (Asynchronous Communications Adapter) input. Waits for a character from the standard auxiliary device, then returns that character in AL.

Notes:

1. Auxiliary (AUX, COM1, COM2) support is unbuffered and non-interrupt driven.

2. At startup, DOS initializes the first auxiliary port to 2400 baud, no parity, one stop bit, and 8-bit word.

3. The auxiliary function calls (3 and 4) do not return status or error codes. For greater control, it is recommended that the ROM BIOS routine (INT hex 14) be used.

4 Auxiliary (Asynchronous Communications Adapter) output. The character in DL is output to the standard auxiliary device.

5 Printer output. The character in DL is output to the standard printer device.

6 Direct console I/O. If DL is hex FF, AL returns with the zero flag clear and an input character from the standard input device if one is ready. If a character is not ready, the zero flag will be set. If DL is not hex FF, then DL is assumed to have a valid character that is output to the standard output device. This function does not check for Ctrl-Break.

7 Direct console input without echo. Waits for a character to be read at the standard input device (unless one is ready), then returns the character in AL. As with function 6, no checks are made on the character.

8 Console input without echo. This function is identical to function 1, except the key is not echoed.

9 Print string. On entry, DS:DX must point to a character string in memory terminated by a $ (hex 24). Each character in the string will be output to the standard output device in the same form as function 2.

A Buffered keyboard input. On entry, DS:DX point to an input buffer. The first byte must not be zero and specifies the number of characters the buffer can hold. Characters are read from the standard input device and placed in the buffer beginning at the third byte. Reading the standard input device and filling the buffer continues until Enter is read. If the buffer fills to one less than the maximum number of characters it can hold, then each additional character read is ignored and causes the bell to ring, until Enter is read. The second byte of the buffer is set to the number of characters received, excluding the carriage return (hex 0D), which is always the last character. Editing of this buffer is described in Chapter 3.

B Check standard input status. If a character is available from the standard input device, AL will be hex FF. Otherwise, AL will be 00. If a Ctrl-Break is detected, an interrupt type hex 23 is executed.

C Clear standard input buffer and invoke a standard input function. Clear the standard input buffer of any pre-typed characters, then execute the function number in AL (only 1, 6, 7, 8, and A are allowed). This forces the system to wait until a character is typed.

D Disk reset. Flushes all file buffers. Files changed in size but not closed are not properly recorded in the disk directory. This function need not be called before a diskette change if all files written have been closed.

E Select disk. The drive specified in DL (0=A, 1=B, etc.) is selected (if valid) as the default drive. The number of drives (total of diskette and fixed disk drives) is returned in AL. If the system has only one diskette drive, it will be counted as two to be consistent with the philosophy of thinking of the system as having logical drives A and B. BIOS equipment determination (INT 11H) can be used as an alternative method, returning the actual number of physical diskette drives.

F Open file. On entry, DS:DX point to a current unopened file control block (FCB). The directory is searched for the named file and AL returns hex FF if it is not found. If it is found, AL returns 00 and the FCB is filled as follows:

If the drive code was 0 (default drive), it is changed to the actual drive used (1=A, 2=B, etc.). This allows changing the default drive without interfering with subsequent operations on this file. The current block field (FCB bytes C-D) is set to zero. The size of the record to be worked with (FCB bytes E-F) is set to the system default of hex 80. The size of the file and the date are set in the FCB from information obtained from the directory.

It is your responsibility to set the record size (FCB bytes E-F) to the size you wish to think of the file in terms of, if the default hex 80 is insufficient. It is also your responsibility to set the random record field and/or current record field. These actions should be done after open but before any disk operations are requested.

10 Close file. This function must be called after file writes to ensure all directory information is updated. On entry, DS:DX point to an opened FCB. The current disk directory is searched and if the file is found, its position is compared with that kept in the FCB. If the file is not found in its correct position in the current directory, it is assumed the diskette was changed and AL returns hex FF. Otherwise, the directory is updated to reflect the status in the FCB and AL returns 00.

11 Search for the first entry. On entry, DS:DX point to an unopened FCB. The current disk directory is searched for the first matching filename (name could have "?"s indicating any letter matches) and if none are found, AL returns hex FF. Otherwise, AL returns 00 and the locations at the disk transfer address are set as follows:

If the FCB provided for searching was an extended FCB, then the first byte at the disk transfer address is set to hex FF followed by five bytes of zeros, then the attribute byte from the search FCB, then the drive number used (1=A, 2=B, etc.), then the 32 bytes of the directory entry. Thus, the disk transfer address contains a valid unopened extended FCB with the same search attributes as the search FCB.

If the FCB provided for searching was a normal FCB, then the first byte is set to the drive number used (1=A, 2=B), and the next 32 bytes contain the matching directory entry. Thus, the disk transfer address contains a valid unopened normal FCB.

Notes:

If an extended FCB is used, the following search pattern is used:

1. If the FCB attribute byte is zero, only normal file entries are found. Entries for volume label, sub-directories, hidden and system files, will not be returned.

2. If the attribute field is set for hidden or system files, or directory entries, it is to be considered as an inclusive search. All normal file entries plus all entries matching the specified attributes are returned. To look at all directory entries except the volume label, the attribute byte may be set to hidden + system + directory (all 3 bits on).

3. If the attribute field is set for the volume label, it is considered an exclusive search, and *only* the volume label entry is returned.

The attribute bits are defined in the "DOS Disk Directory" section of Appendix C.

12 Search for the next entry. After function 11 has been called and found a match, function 12 may be called to find the next match to an ambiguous request (?s in the search filename). Both inputs and outputs are the same as function 11. The reserved area of the FCB keeps information necessary for continuing the search, so no disk operations may be performed with this FCB between a previous function 11 or 12 call and this one.

13 Delete file. On entry, DS:DX point to an unopened FCB. All matching current directory entries are deleted. If no directory entries match, AL returns hex FF; otherwise AL returns 00.

14 Sequential read. On entry, DS:DX point to an opened FCB. The record addressed by the current block (FCB bytes C-D) and the current record (FCB byte 1F) is loaded at the disk transfer address, then the record address is incremented. (The length of the record is determined by the FCB record size field.) If end-of-file is encountered, AL returns either 01 or 03. A return of 01 indicates no data in the record; 03 indicates a partial record is read and filled out with zeros. A return of 02 means there was not enough space in the disk transfer segment to read one record, so the transfer was ended. AL returns 00 if the transfer was completed successfully.

15 Sequential write. On entry, DS:DX point to an opened FCB. The record addressed by the current block and current record fields (size determined by the FCB record size field) is written from the disk transfer address (or, in the case of records less than sector sizes, is buffered up for an eventual write when a sector's worth of data is accumulated). The record address is then incremented. If the diskette is full, AL returns 01. A return of 02 means there was not enough space in the disk transfer segment to write one record, so the transfer was ended. AL returns 00 if the transfer was completed successfully.

16 Create file. On entry, DS:DX point to an unopened FCB. The current disk directory is searched for a matching entry, and if found, it is re-used. If no match was found, the directory is searched for an empty entry, and AL returns FF if none is found. Otherwise, the entry is initialized to a zero-length file, the file is opened (see function F), and AL returns 00.

The file may be marked *hidden* during its creation by using an extended FCB containing the appropriate attribute byte.

17 Rename file. On entry, DS:DX point to a modified FCB which has a drive code and a file name in the usual position, and a second file name starting 6 bytes after the first (DS:DX+hex 11) in what is normally a reserved area. Every matching occurrence of the first name in the current directory is changed to the second (with the restriction that two files cannot have the same name and extension). If "?"s appear in the second name, then the corresponding positions in the original name will be unchanged. AL returns FF if no match was found or if an attempt was made to rename to a filename that already existed, otherwise 00.

18 Used internally by DOS.

19 Current disk. AL returns with the code of the current default drive (0=A, 1=B, etc.).

1A Set disk transfer address. The disk transfer address is set to DS:DX. DOS does not allow disk transfers to wrap around within the segment, or overflow into the next segment.

1B Allocation table information. On return, DS:BX point to a byte containing the FAT identification byte for the default drive, DX has the number of allocation units, AL has the number of sectors per allocation unit, and CX has the size of the physical sector.

> **Note:** Beginning with DOS Version 2.0, this call no longer returns the address of a complete File Allocation Table, because the FAT's are no longer kept resident in memory.

1C Allocation table information for specific drive. This call is identical to call hex 1B except that on entry, DL contains the number of the drive from which the information should be gotten (0 = default, 1= A, etc.).

1D Used internally by DOS

1E Used internally by DOS

1F Used internally by DOS

20 Used internally by DOS

21 Random read. On entry, DS:DX point to an opened FCB. The current block and current record fields are set to agree with the random record field, then the record addressed by these fields is read into memory at the current disk transfer address. If end-of-file is encountered, AL returns either 01 or 03. If 01 is returned, no more data is available. If 03 is returned, a partial record is available filled out with zeros. A return of 02 means there was not enough space in the disk transfer segment to read one record, so the transfer was ended. AL returns 00 if the transfer was completed successfully.

22 Random write. On entry, DS:DX point to an opened FCB. The current block and current record fields are set to agree with the random record field, then the record addressed by these fields is written (or in the case of records not the same as sector sizes - buffered) from the disk transfer address. If the diskette is full AL returns to 01. A return of 02 means there was not enough space in the disk transfer segment to write one record; so, the transfer was ended. AL returns 00 if the transfer was completed successfully.

23 File size. On entry, DS:DX point to an unopened FCB. The diskette directory is searched for the first matching entry and if none is found, AL returns FF. Otherwise, the random record field is set to the number of records in the file (in terms of the record size field rounded up) and AL returns 00.

> **Note:** Be sure to set the FCB record size field before using this function call; otherwise, erroneous information will be returned.

24 Set random record field. On entry, DS:DX point to an opened FCB. This function sets the random record field to the same file address as the current block and record fields.

25 Set interrupt vector. The interrupt vector table for the interrupt type specified in AL is set to the 4-byte address contained in DS:DX. Note that the original contents of the interrupt vector can be obtained through call hex 35.

26 Create a new program segment. On entry, DX has a segment number at which to set up a new program segment. The entire hex 100 area at location zero in the current program segment is copied into location zero in the new program segment. The memory size information at location 6 in the new segment is updated and the current termination, Ctrl-Break exit and critical error addresses from interrupt vector table entries for interrupt types 22, 23, and 24 are saved in the new program segment starting at hex 0A. They are restored from this area when the program terminates.

> **Note:** Use of this call should be avoided, now that DOS contains the EXEC function call (hex 4B).

27 Random block read. On entry, DS:DX point to an opened FCB, and CX contains a record count that must not be zero. The specified number of records (in terms of the record size field) are read from the file address specified by the random record field into the disk transfer address. If end-of-file is reached before all records have been read, AL returns either 01 or 03. A return of 01 indicates end-of-file and the last record is complete. A return of 03 indicates the last record is a partial record. If wrap-around above address hex FFFF in the disk transfer segment would have occurred, as many records as possible are read and AL returns 02. If all records are read successfully, AL returns 00. In any case, CX returns with the actual number of records read, and the random record field and the current block/record fields are set to address the next record (the first record not read).

28 Random block write. Essentially the same as function 27 above, except for writing and a write-protect check. If there is insufficient space on the disk, AL returns 01 and no records are written. If CX is zero upon entry, no records are written, but the file is set to the length specified by the random record field, whether longer or shorter than the current file size. (Allocation units are released or allocated as appropriate.)

29 Parse filename. On entry, DS:SI point to a command line to parse, and ES:DI point to a portion of memory to be filled with an unopened FCB. The contents of AL are used to determine the action to take, as shown below:

```
        <ignored>
bit:  7  6  5  4  3  2  1  0
```

If bit 0 = 1, then leading separators are scanned off the command line at DS:SI. Otherwise, no scan-off of leading separators takes place.

If bit 1 = 1, then the drive ID byte in the result FCB will be set (changed) *only* if a drive was specified in the command line being parsed.

If bit 2 = 1, then the filename in the FCB will be changed only if the command line contains a filename.

If bit 3 = 1, then the filename extension in the FCB will be changed only if the command line contains a filename extension.

Filename separators include the following characters: . ; , = + plus TAB and SPACE. Filename terminators include all of these characters plus \, <, >, | , /, ", [,], and any control characters.

The command line is parsed for a filename of the form d:filename.ext, and if found, a corresponding unopened FCB is created at ES:DI. If no drive specifier is present, the default drive is assumed. If no extension is present, it is assumed to be all blanks. If the character * appears in the filename or extension, then it and all remaining characters in the name or extension are set to ?.

If either ? or * appears in the filename or extension, AL returns 01; if the drive specifier is invalid AL returns FF; otherwise 00.

DS:SI will return pointing to the first character after the filename and ES:DI will point to the first byte of the formatted FCB. If no valid filename is present, ES:DI+1 will contain a blank.

> Note: This call is not useful for command lines containing path names.

2A Get date. Returns date in CX:DX. CX has the year (1980-2099 in binary), DH has the month (1-Jan, 2-Feb, etc) and DL has the day. If the time-of-day clock rolls over to the next day, the date is adjusted accordingly, taking into account the number of days in each month and leap years.

2B Set date. On entry, CX:DX must have a valid date in the same format as returned by function 2A, above. If the date is indeed valid and the set operation is successful, AL returns 00. If the date is not valid, AL returns FF.

2C Get time. Returns with time-of-day in CX:DX. Time is actually represented as four 8-bit binary quantities as follows. CH has the hours (0-23), CL has minutes (0-59), DH has seconds (0-59), DL has 1/100 seconds (0-99). This format is readily converted to a printable form yet can also be used for calculations, such as subtracting one time value from another.

2D Set time. On entry, CX:DX has time in the same format as returned by function 2C, above. If any component of the time is not valid, the set operation is aborted and AL returns FF. If the time is valid, AL returns 00.

2E Set/reset verify switch. On entry, DL must contain 0, and AL must contain 1 to turn verify on, or 0 to turn verify off. When on, DOS will perform a verify operation each time it performs a diskette write to assure proper data recording. Although disk recording errors are very rare, this function has been provided for those user applications in which you may wish to verify the proper recording of critical data. Note that the current setting of the verify switch can be obtained through call hex 54.

2F Get DTA. On return, ES:BX contains the current DTA transfer address.

30 Set DOS version number. On return, AL contains the major version number. AH contains the minor version number.

> **Note:** If AL returns zero, it can be assumed that it is a pre-DOS Version 2.00 system.

31 Terminates process and remain resident (KEEP process). On entry, AL contains a binary exit code. DX contains the memory size value in paragraphs. This function call terminates the current process and attempts to set the initial allocation block to the number of paragraphs in DX. It will not free up any other allocation blocks belonging to that process. The exit code passed in AL is retrievable by the parent through Wait (function call hex 4D 0) and can be tested through the ERRORLEVEL batch subcommands.

32 Used internally by DOS.

33 Ctrl-Break check. On entry, AL contains 00 to request the current state of control-break checking, 01 to set the state. If setting the state, DL must contain 00 for OFF or 01 for ON. DL returns the current state (00 = OFF, 01 = ON).

34 Used internally by DOS.

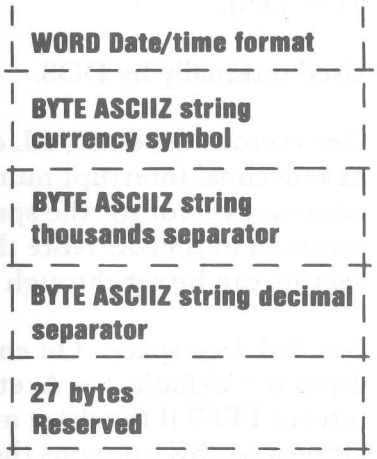
35 Get vector. On entry, AL contains a hexadecimal interrupt number. The CS:IP interrupt vector for the specified interrupt is returned in ES:BX. Note that interrupt vectors can be set through call hex 25.

36 Get disk free space. On entry, DL contains a drive: 0 = default, 1 = A, etc,. On return, AX returns FFFF if the drive number was invalid. Otherwise, BX contains the number of available allocation units (clusters), DX contains the total number of clusters on the drive, CX contains the number of bytes per sector, and AX contains the number of sectors per cluster.

> **Note:** This call returns the same information in the same registers (except for the FAT pointer) as the get FAT pointer call (hex 1B) did in previous versions of DOS.

37 Used internally by DOS.

38 Return country dependent information (international). On entry, DS:DX points to a 32-byte block of memory in which returned information is passed and AL contains a function code. In DOS 2.00, this function code must be zero. The following information is pertinent to international applications.



The date and time format has the following values and meaning:

0 = USA standard *h:m:s m/d/y*

1 = Europe standard *h:m:s d/m/y*

2 = Japan standard *h:m:s d/m/y*

39 Create a sub-directory (MKDIR). On entry, DS:DX contains the address of an ASCIIZ string with drive and directory path names. If any member of the directory path does not exist, then the directory path is not changed. On return, a new directory is created at the end of the specified path. Error returns are 3 and 5 (refer to error return table).

3A Remove a directory entry (RMDIR). On entry, DS:DX contains the address of an ASCIIZ string with the drive and directory path names. The specified directory is removed from the structure. The current directory cannot be removed. Error returns are 3 and 5 (refer to error return table). Note that code 5 is returned if the specified directory is not empty.

3B Change the current directory (CHDIR). On entry, DS:DX contains the address of an ASCIIZ string with drive and directory path names. If any member of the directory path does not exist, then the directory path is not changed. Otherwise, the current directory is set to the ASCIIZ string. Error return is 3 (refer to the error return table).

3C Create a file (CREAT). On entry, DS:DX contains the address of an ASCIIZ string with the drive, path, and filename. CX contains the attribute of the file. This function call creates a new file or truncates an old file to zero length in preparation for writing. If the file did not exist, then the file is created in the appropriate directory and the file is given the read/write access code. The file is opened for read/write, and the handle is returned in AX. Error returns are 3, 4, and 5 (refer to the error return table). If an error code of 5 is returned, either the directory was full or a file by the same name exists and is marked read-only. Note that the change mode function call (hex 43) can later be used to change the file's attribute.

3D Open a file. On entry, DS:DX contains the address of an ASCIIZ string with the drive, path, and filenames. AL contains the access code. On return, AX contains an error code or a 16-bit file handle associated with the file. The following values are allowed for the access code:

0 = file is opened for reading.

1 = file is opened for writing.

2 = file is opened for both reading and writing.

The read/write pointer is set at the first byte of the file and the record size of the file is 1 byte (the read/write pointer can be changed through function call hex 42). The returned file handle must be used for subsequent input and output to the file. The file's date and time can be obtained or set through call hex 57, and its attribute can be obtained through call hex 43. Error returns are 2, 4, 5, and 12 (refer to the error return table).

Note: This call will open any normal or hidden file whose name matches the name specified.

3E Close a file handle. On entry, BX contains the file handle that was returned by "open." On return, the file will be closed and all internal buffers are flushed. Error return is 6 (refer to the error return table).

3F Read from a file or device. On entry, BX contains the file handle. CX contains the number of bytes to read. DS:DX contains the buffer address. On return, AX contains the number of bytes read. If the value is zero, then the program has tried to read from the end of file. This function call transfers (CX) bytes from a file into a buffer location. It is not guaranteed that all bytes will be read. For example, reading from the keyboard will read at most one line of text. If this read is performed from the standard input device, the input can be redirected (see "Redirection of Standard Input and Output" in Chapter 10). Error returns are 5 and 6 (refer to the error return table).

40 Write to a file or device. On entry, BX contains the file handle. CX contains the number of bytes to write. DS:DX contains the address of the data to write. Write transfers (CX) bytes from a buffer into a file. AX returns the number of bytes actually written. If this value is not the same as the number requested, it should be considered an error (no error code is returned, but your program can compare these values). The usual reason for this is a full disk. If this write is performed to the standard output device, the output can be redirected (see "Redirection of Standard Input and Output" in Chapter 10). Error returns are 5 and 6 (refer to the error return table).

41 Delete a file from a specified directory (Unlink). On entry, DS:DX contains the address of an ASCIIZ string with a drive, path, and filename. Global filename characters are not allowed in any part of the string. This function call removes a directory entry associated with a filename. Read-only files cannot be deleted by this call. To delete one of these files, you can first use call hex 43 to change the file's attribute to 0, then delete the file. Error returns are 2 and 5 (refer to the error return table).

42 Move file read/write pointer (Lseek). On entry, AL contains a method value. BX contains the file handle. CX:DX contains the desired offset in bytes (CX contains the most significant part). On return, DX:AX contains the new location of the pointer (DX contains the most significant part).

It moves the read/write pointer according to the following methods:

AL 0 = The pointer is moved to offset (CX:DX) bytes from the beginning of the file.

AL 1 = The pointer is moved to the current location plus offset.

AL 2 = The pointer is moved to the end-of-file plus offset. This method can be used to determine file's size.

Error returns are 1 and 6 (refer to the error return table).

43 Change file mode (CHMOD). On entry, AL contains a function code, and DS:DX contains the address of an ASCIIZ string with the drive, path, and filename. If AL contains 01 then the file will be set to the attribute in CX. (See "DOS Disk Directory" in Appendix C for the attribute byte description.) If AL is 0 then the file's current attribute will be returned in CX. Error returns are 3 and 5 (refer to the error return table).

44 I/O control for devices (IOCTL). On entry, AL contains the function value. BX contains the file handle. On return, AX contains the number of bytes transferred for functions 2, 3, 4, and 5 or status (00 = not ready, FF = ready) for functions 6 and 7, or an error code. Use IOCTL to Set or Get device information associated with open device handle, or send/receive control strings to the device handle. The following function values are allowed in AL:

0 = Get device information (returned in DX).

1 = Set device information (determined by DX). Currently, DH must be zero for this call.

2 = Read CX number of bytes into DS:DX from device control channel.

3 = Write CX number of bytes from DS:DX to device control channel.

4 = Same as 2, but use drive number in BL (0 = default, 1 = A, etc.).

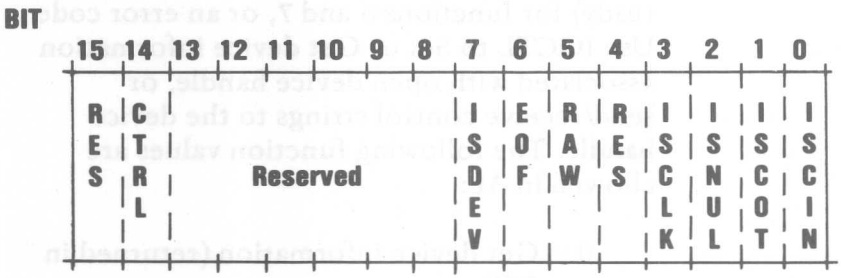
5 = Same as 3, but use drive number in BL (0 = default, 1 = A, etc.).

6 = Get input status.

7 = Get output status.

IOCTL can be used to get information about device channels. You can make calls on regular files, but only function values 0, 6, and 7 are defined in that case. All other calls return an "invalid function" error.

Calls AL=0 and AL=1. The bits of DX are defined as follows:



ISDEV = 1 if this channel is a device.
0 if this channel is a disk file (bits 8-15 = 0 in this case).

If ISDEV = 1

EOF = 0 if end-of-file on input.

BIN = 1 if operating in binary mode (no checks for Ctrl-Z).
= 0 if operating in ASCII mode (checking for Ctrl-Z as end-of-file).

ISCLK = 1 if this device is the clock device.

ISNUL = 1 if this device is the null device.

ISCOT = 1 if this device is the console output.

ISCIN = if this device is the console input.

CTRL = 0 if this device cannot process control strings via calls AL=2 and AL=3.

CTRL = 1 if this device can process control strings via calls AL=2 and AL=3. Note that this bit can not be set by function call hex 44.

If ISDEV = 0

EOF = 0 if channel has been written. Bits 0-5 are the block device number for the channel (0 = A, 1 = B, ...).

Bits 15, 8-13, 4 are reserved and should not be altered.

Note: DH must be zero for call AL=1.

Calls AL=2, AL=3, AL=4, AL=5. These four calls allow arbitrary control strings to be sent or received from a character device. The Call syntax is the same as the Read and Write calls, except for calls 4 and 5 which accept a drive number in BL instead of a handle in BX. An "invalid function" error is returned if the CTRL bit is zero. An "access-denied" code is returned by calls 4 and 5 if the drive is invalid. Error returns are 1, 6, and 13 (refer to the error return table).

Calls 6 and 7. These calls allow you to check if a file handle is ready for input or output. If used for a file, AL always returns FF until end-of-file is reached, then always returns 00 unless the current file position is changed through call hex 42. When used for a device, AL returns FF for ready or zero for not ready.

45 Duplicate a file handle (DUP). On entry, BX contains the file handle. On return, AX contains the returned file handle. This function call takes an opened file handle and returns a new file handle that refers to the same file at the same position. Error returns are 4 and 6 (refer to the error return table).

> **Note:** If you move the read/write pointer of either handle, the pointer for the other handle will also be changed.

46 Force a duplicate of a handle (DUP). On entry, BX contains the file handle. CX contains a second file handle. On return, the CX file handle will refer to the same stream as the BX file handle. If the CX file handle was an open file, then it is closed first. Error return is 6 (refer to the error return table).

> **Note:** If you move the read/write pointer of either handle, the pointer for the other handle will also be changed.

47 Get Current directory. On entry, DL contains a drive number (0 = default, 1 = A, etc.) and DS:SI point to a 64-byte area of user memory. The full path name (starting from the root directory) of the current directory for the specified drive is placed in the area pointed to by DS:SI. Note that the drive letter will not be part of the returned string. The string will not begin with a backslash and will be terminated by a byte containing hex 00. The error returned is 15.

48 Allocate memory. On entry, BX contains the number of paragraphs requested. On return, AX:0 points to the allocated memory block. If the allocation fails, BX will return the size of the largest block of memory available in paragraphs. Error returns are 7 and 8 (refer to the error return table).

49 Free allocated memory. On entry, ES contains the segment of the block being returned. On return, a block of memory is returned to the system pool that was allocated by call hex 48. Error returns are 7 and 9 (refer to the error return table).

4A SETBLOCK-Modify allocated memory blocks. On entry, ES contains the segment of the block. BX contains the new requested block size in paragraphs. DOS will attempt to "grow" or "shrink" the specified block. If the call fails on a grow request, then on return, BX contains the maximum block size possible. Error returns are 7, 8, and 9 (refer to the error return table).

4B Load or execute a program (Exec). This function call allows a program to load another program into memory and (default) begin execution of it. DS:DX points to the ASCIIZ string with drive, path, and filename of the file to be loaded. ES:BX points to a parameter block for the load and AL contains a function value. The following function values are allowed:

0 = Load and execute the program. A program segment prefix is established for the program and the terminate and control-break addresses are set to the instruction after the EXEC system call.

Note: When control is returned, all registers are changed including the stack. You must restore SS, SP and any other required registers before proceeding.

3 = Load, do not create the program segment prefix, and do not begin execution. This is useful in loading program overlays.

For each of these values, the block pointed to by ES:BX has the following format:

AL = 0 Load/execute program

WORD segment address of environment string to be passed.
DWORD pointer to command line to be placed at PSP+ 80h
DWORD points to default FCB to be passed at PSP+ 5Ch
DWORD pointer to default FCB to be passed at PSP+ 6Ch

AL = 3 Load overlay

WORD segment address where file will be loaded
WORD relocation factor to be applied to the image

Note that all open files of a process are duplicated in the newly created process after an Exec. This is extremely powerful; the parent process has control over the meanings of standard input, output, auxiliary, and printer devices. The parent could, for example, write a series of records to a file, open the file as standard input, open a listing file as standard output, and then execute a sort program that takes its input from standard input and writes to standard output.

Also inherited (or copied from the parent) is an "environment." This is a block of text strings (less than 32K bytes total) that convey various configuration parameters. The following is the format of the environment (always on a paragraph boundary):

Byte ASCIIZ string 1
Byte ASCIIZ string 2
...
Byte ASCIIZ string n
Byte of zero

Typically the environment strings have the form:

parameter=value

For example, the string VERIFY=ON could be passed. A zero value of the environment address will cause the newly created process to inherit the parent's environment unchanged. The segment address of the environment is placed at offset hex 2C of the Program Segment Prefix for the program being invoked. Error returns are 1, 2, 5, 8, 10, and 11 (refer to the error return table).

Notes:

1. When your program received control, all of available memory was allocated to it. You must free some memory (see call hex 4A) before EXEC can load the program you are invoking. Normally, you would shrink down to the minimum amount of memory you need, and free the rest.

2. The EXEC call uses the loader portion of COMMAND.COM (at the high end of memory) to perform the loading. If your program has overlaid the loader, this call will attempt to re-load the loader, thus destroying the contents of the last 1536 bytes of memory. If you have used the "Allocate Memory" call to allocate all of memory and the loader has been overlaid, the EXEC call will return an error due to insufficient memory to load the loader.

4C Terminate a process (Exit). On entry, AL contains a binary return code. This function call will terminate the current process, transferring control to the invoking process. In addition, a return code can be sent. The return code can be interrogated by the batch subcommands IF and ERRORLEVEL and by the wait function call (4D). All files open at the time are closed.

4D Retrieve the return code of a sub-process (Wait). This function call returns the Exit code specified by another process (via call hex 4C or call hex 31) in AX. It returns the Exit code only once. The low byte of this code is that sent by the exiting routine. The high byte is zero for normal termination, 01 if terminated by Crtl-Break, 02 if terminated as the result of a critical device error, or 03 if terminated by function call hex 31.

4E Find first matching file (FIND FIRST). On input, DS:DX points to an ASCIIZ string containing the drive, path, and filename of the file to be found. The filename portion can contain global filename characters. CX contains the attribute to be used in searching for the file. See function call hex 11 for a description of how the attribute bits are used for searches. If a file is found that matches the specified drive, path, and filename and attribute, the current DTA will be filled in as follows:

21 bytes - reserved for DOS use on subsequent find next calls

1 byte - attribute found

2 bytes - file's time

2 bytes - file's date

2 bytes - low word of file size

2 bytes - high word of file size

13 bytes - name and extension of file found, followed by a byte of zeros. All blanks are removed from the name and extension, and if an extension is present, it is preceded by a period. Thus, the name returned appears just as you would enter it as a command parameter. Such as, TREE.COM followed by a byte of zeros. Error returns are 2 and 18 (refer to the error return table).

4F Find next matching file. On input, the current DTA must contain the information that was filled in by a previous Find First call (hex 4E). No other input is required. This call will find the next directory entry matching the name that was specified on the previous Find First call. If a matching file is found, the current DTA will be set as described in call hex 4E. If no more matching files are found, error code 18 is returned (refer to the error return table).

50 Used internally by DOS.

51 Used internally by DOS.

52 Used internally by DOS.

53 Used internally by DOS.

54 Get verify state. On return, AL returns 00 if verify is OFF, 01 if verify is ON. Note that the verify switch can be set through call hex 2E.

55 Used internally by DOS.

56 Rename a file. On input, DS:DX points to an ASCIIZ string containing the drive, path, and filename of the file to be renamed. ES:DI points to an ASCIIZ string containing the path and filename to which the file is to be renamed. If a drive is used in this string, it must be the same as the drive specified or implied in the first string. The directory paths need not be the same, allowing a file to be moved to another directory and renamed in the process. Error returns are 3, 5, and 17 (refer to the error return table).

57 Get/Set a file's date and time. On input, AL contains 00 or 01. BX contains a file handle. If AL=00 on entry, DX and CX will return the date and time from the handle's internal table, respectively. If AL=01 on entry, the handle's date and time will be set to the date and time in DX and CX, respectively. The date and time formats are the same as those for the directory entry described in Appendix C, except that when passed in registers, the bytes are reversed (that is, DH contains the *low* order portion of the date, etc.). Error returns are 1 and 6 (refer to the error return table).

Notes:

Appendix E. DOS Control Blocks and Work Areas

DOS Memory Map

0000:0000	Interrupt vector table
0040:0000	ROM communication area
0050:0000	DOS communication area
XXXX:0000	IBMBIO.COM — DOS interface to ROM I/O routines
XXXX:0000	IBMDOS.COM — DOS interrupt handlers, service routines (INT 21 functions)
	DOS buffers, control areas, and installed device drivers.
XXXX:0000	Resident portion of COMMAND.COM — Interrupt handlers for interrupts hex 22 (terminate), hex 23 (Ctrl-Break), hex 24 (critical error), and code to reload the transient portion.
XXXX:0000	External command or utility — (.COM or .EXE file)
XXXX:0000	User stack for .COM files (256 bytes)
XXXX:0000	Transient portion of COMMAND.COM — Command interpreter, internal commands, batch processor, external command loader.

Notes:

1. Memory map addresses are in segment:offset format. For example, 0070:0000 is absolute address hex 0700.

2. The DOS Communication Area is used as follows:

0050:0000	Print screen status flag store	
	0	Print screen not active or successful print screen operation
	1	Print screen in progress
	255	Error encountered during print screen operation
0050:0001	Used by BASIC	
0050:0004	Single-drive mode status byte	
	0	Diskette for drive A was last used
	1	Diskette for drive B was last used
0050:0010 – 0021	Used by BASIC	
0050:0022 – 002F	Used by DOS for diskette initialization	
0050:0030 – 0033	Used by MODE command	

All other locations within the 256 bytes beginning at 0050:0000 are reserved for DOS use.

3. User memory is allocated from the lowest end of available memory that will satisfy the request for memory.

DOS Program Segment

When you enter an external command, or invoke a program through the EXEC function call, DOS determines the lowest available address to use as the start of available memory for the program being invoked. This area is called the Program Segment (it must not be moved).

At offset 0 within the Program Segment, DOS builds the Program Segment Prefix control block. (See below.) EXEC loads the program at offset hex 100 and gives it control.

The program returns from EXEC by a jump to offset 0 in the Program Segment Prefix, by issuing an INT 20, by issuing an INT 21 with register AH=0 or hex 4C, or by calling location hex 50 in the Program Segment Prefix with AH=0 or hex 4C.

> **Note:** It is the responsibility of all programs to ensure that the CS register contains the segment address of the Program Segment Prefix when terminating via any of these methods except call hex 4C.

All four methods result in transferring control to the resident portion of COMMAND.COM (function call hex 4C allows the terminating process to pass a return code). All of these methods result in turning to the program that issued the EXEC. During this returning process, interrupt vectors hex 22, hex 23, and hex 24 (terminate, Ctrl-Break, and critical error exit addresses) are restored from the values saved in the Program Segment Prefix of the terminating program. Control is then given to the terminate address. If this is a program returning to COMMAND, control transfers to its transient portion. If a batch file was in process, it is continued; otherwise, COMMAND issues the system prompt and waits for the next command to be entered from the keyboard.

When a program receives control, the following conditions are in effect:

For all programs:

- The segment address of the passed environment is contained at offset hex 2C in the Program Segment Prefix.

  The environment is a series of ASCII strings (totaling less than 32K) in the form:

  NAME=parameter

Each string is terminated by a byte of zeros, and the entire set of strings is terminated by another byte of zeros. The environment built by the command processor (and passed to all programs it invokes) will contain a COMSPEC= string at a minimum (the parameter on COMSPEC is the path used by DOS to locate COMMAND.COM on disk). The last PATH and PROMPT commands issued will also be in the environment, along with any environment strings entered through the SET command (see Chapter 10).

The environment that you are passed is actually a copy of the invoking process environment. If your application uses a "terminate and stay resident" concept, you should be aware that the copy of the environment passed to you is static. That is, it will not change even if subsequent SET, PATH, or PROMPT commands are issued.

- Offset hex 50 in the Program Segment Prefix contains code to invoke the DOS function dispatcher. Thus, by placing the desired function number in AH, a program can issue a long call to PSP +50 to invoke a DOS function, rather than issuing an interrupt type hex 21.
- Disk transfer address (DTA) is set to hex 80 (default DTA in the Program Segment Prefix).
- File control blocks at hex 5C and hex 6C are formatted from the first two parameters entered when the command was invoked. Note if either parameter contained a path name, then the corresponding FCB will contain only a valid drive number. The filename field will not be valid.

- An Unformatted parameter area at hex 81 contains all the characters entered after the command name (including leading and imbedded delimiters), with hex 80 set to the number of characters. If the <, >, or parameters were entered on the command line, they (and the filenames associated with them) will not appear in this area, because redirection of standard input and output is transparent to applications.

- Offset 6 (one word) contains the number of bytes available in the segment.

- Register AX reflects the validity of drive specifiers entered with the first two parameters as follows:

  — AL=FF if the first parameter contained an invalid drive specifier (otherwise AL=00)

  — AH=FF if the second parameter contained an invalid drive specifier (otherwise AH=00)

For .EXE programs:

- DS and ES registers are set to point to the Program Segment Prefix. (See below.)

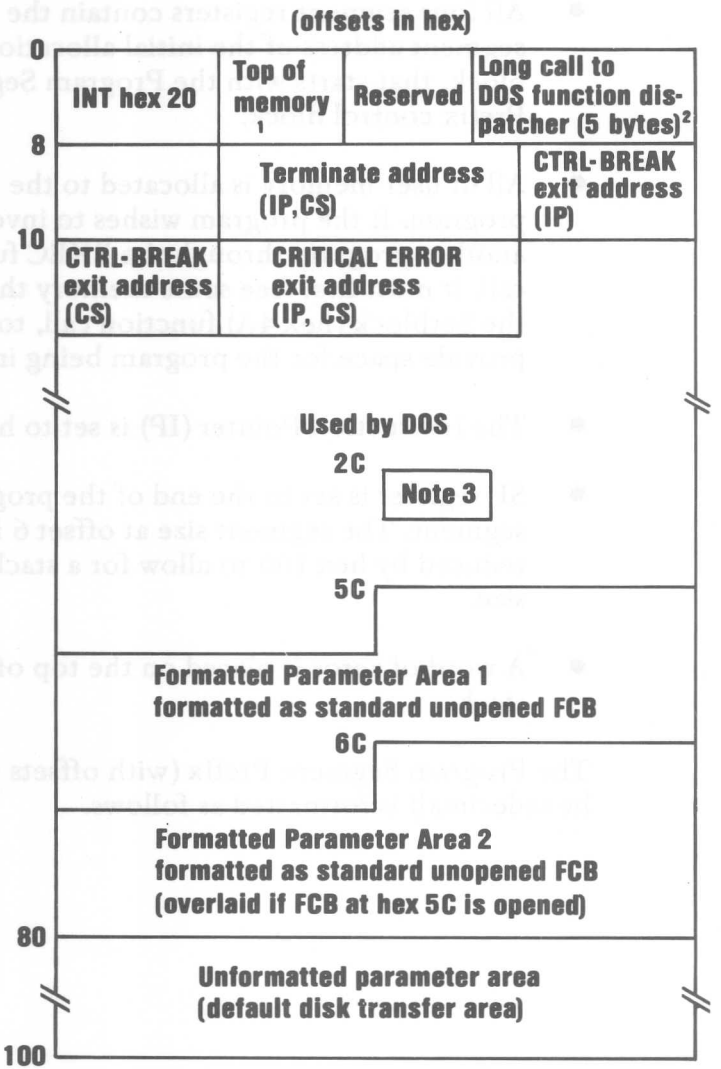
- CS, IP, SS, and SP registers are set to the values passed by the linker.

For .COM programs:

- All four segment registers contain the segment address of the initial allocation block, that starts with the Program Segment Prefix control block.
- All of user memory is allocated to the program. If the program wishes to invoke another program through the EXEC function call, it must first free some memory through the Setblock (hex 4A) function call, to provide space for the program being invoked.
- The Instruction Pointer (IP) is set to hex 100.
- SP register is set to the end of the program's segment. The segment size at offset 6 is reduced by hex 100 to allow for a stack of that size.
- A word of zeros is placed on the top of the stack.

The Program Segment Prefix (with offsets in hexadecimal) is formatted as follows.

Program Segment Prefix



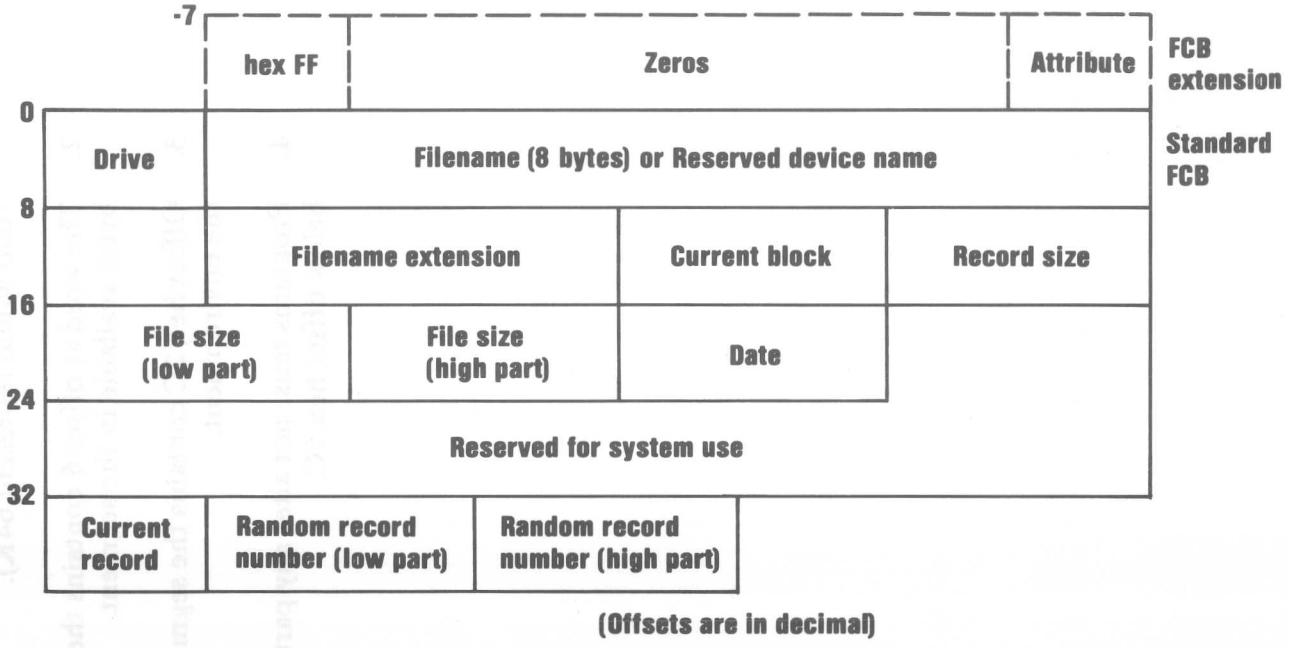
1. First segment of available memory is in segment (paragraph) form (for example, hex 1000 would represent 64K).

2. The word at offset 6 contains the number of bytes available in the segment.

3. Offset hex 2C contains the segment address of the environment.

4. Programs must not alter any part of the PSP below offset hex 5C.

File Control Block



Unshaded areas must be filled in by the using program.

Shaded areas are filled in by DOS and must not be modified.

Standard File Control Block

The standard file control block (FCB) is defined as follows, with the offsets in decimal:

Byte	Function
0	Drive number. For example, **Before open:** 0 - default drive 1 - drive A 2 - drive B etc. **After open:** 1 - drive A 2 - drive B etc. A 0 is replaced by the actual drive number during open.
1-8	Filename, left-justified with trailing blanks. If a reserved device name is placed here (such as LPT1), do not include the optional colon.
9-11	Filename extension, left-justified with trailing blanks (can be all blanks).
12-13	Current block number relative to the beginning of the file, starting with zero (set to zero by the open function call). A block consists of 128 records, each of the size specified in the logical record size field. The current block number is used with the current record field (below) for sequential reads and writes.

14-15 Logical record size in bytes. Set to hex 80 by the open function call. If this is not correct, you must set the value because DOS uses it to determine the proper locations in the file for all disk reads and writes.

16-19 File size in bytes. In this 2-word field, the first word is the low-order part of the size.

20-21 Date the file was created or last updated. The *mm/dd/yy* are mapped in the bits as follows:

<							21								>	<																>
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																
y	*y*	*y*	*y*	*y*	*y*	*y*	*m*	*m*	*m*	*m*	*d*	*d*	*d*	*d*	*d*																

where:

mm is 1-12
dd os 1-31
yy is 0-119 (1980-2099)

22-31 Reserved for system use.

32 Current relative record number (0-127) within the current block. (See above.) You must set this field before doing *sequential* read/write operations to the diskette. (This field is not initialized by the open function call.)

33-36 Relative record number relative to the beginning of the file, starting with zero. You must set this field before doing *random* read/write operations to the diskette. (This field is not initialized by the open function call.)

If the record size is less than 64 bytes, both words are used. Otherwise, only the first three bytes are used. Note that if you use the File Control Block at hex 5C in the program segment, the last byte of the FCB overlaps the first byte of the unformatted parameter area.

Notes:

1. An unopened FCB consists of the FCB prefix (if used), drive number, and filename/extensions properly filled in. An open FCB is one in which the remaining fields have been filled in by the Create or Open function calls.

2. Bytes 0-15 and 32-36 must be set by the user program. Bytes 16-31 are set by DOS and must not be changed by user programs.

3. All word fields are stored with the least significant byte first. For example, a record length of 128 is stored as hex 80 at offset 14, and hex 00 at offset 15.

Extended File Control Block

The extended File Control Block is used to create or search for files in the disk directory that have special attributes.

It adds a 7-byte prefix to the FCB, formatted as follows:

Byte	Function
FCB-7	Flag byte containing hex FF to indicate an extended FCB.
FCB-6 to FCB-2	Reserved.
FCB-1	Attribute byte. See "DOS Disk Directory" in Appendix C for attribute bit definitions. Also refer to function call hex 11 (search first) for details on using the attribute bits during directory searches. This function is present to allow applications to define their own files as *hidden* (and thereby exclude them from directory searches), and to allow selective directory searches.

Any references in the DOS Function Calls (refer to Appendix D) to an FCB, whether opened or unopened, may use either a normal or extended FCB. If using an extended FCB, the appropriate register should be set to the first byte of the prefix, rather than the drive-number field.

Appendix F. Executing Commands from Within an Application

Beginning with DOS Version 2.00, application programs may invoke a secondary copy of the command processor. Your program may pass a DOS command as a parameter, that the secondary command processor will execute as though it had been entered from the standard input device. The procedure is:

1. Assure that adequate free memory (17K) exists to contain the second copy of the command processor and the command it is to execute.

2. Build a parameter string for the secondary command processor in the form:

   1 byte = length of parameter string
   xx byte = parameter string
   1 byte = hex 0D (carriage return)

   For example, the assembly statement below would build the string to cause execution of a DISKCOPY command:

```
DB 17,"/C DISKCOPY A: B:", 13
```

3. Use the EXEC function call (hex 4B, function value 0) to cause execution of the secondary copy of the command processor (the drive, directory and name of the command processor can be gotten from the COMSPEC=parameter in the environment passed to you at PSP+X'2C'). Remember to set offset 2 of the EXEC control block to point to the parameter string built above.

Refer to Chapter 10, "Advanced Commands" for additional information about invoking a second copy of the command processor.

Appendix G. Fixed Disk Information

The IBM Personal Computer Fixed Disk Support Architecture has been designed to meet the following objectives:

- Allow multiple operating systems to utilize the fixed disk without the need to dump/restore when changing operating systems.
- Allow a user-selected operating system to be started from the fixed disk.

Fixed Disk Architecture

The architecture is defined as follows

- In order to *share* the fixed disk among operating systems, the disk may be logically divided into 1 to 4 "partitions." The space within a given partition is contiguous, and can be dedicated to a specific operating system. Each operating system may "own" only one partition. The number and sizes of the partitions is user-selectable through a fixed disk utility program. (The DOS utility is FDISK.COM.) The partition information is kept in a partition table that is imbedded in the master fixed disk boot record on the first sector of the disk.

- Any operating system must consider its partition to be an entire disk, and must ensure that its functions and utilities do not access other partitions on the disk.

- Each partition can contain a boot record on its first sector, and any other programs or data that you choose—including a copy of an operating system. For example, the DOS FORMAT command may be used to format (and place a copy of DOS in) the DOS partition, in the same manner that a diskette is formatted. With the FDISK utility, you may designate a partition as "bootable" (active)—the master fixed disk boot record will cause that partition's boot record to receive control when the system is started or restarted.

System Initialization

The System initialization (or system boot) sequence is as follows:

1. System initialization first attempts to load an operating system from diskette drive A. If the drive is not ready or a read error occurs, it then attempts to read a master fixed disk boot record from the first sector of the first fixed disk on the system. If unsuccessful, or if no fixed disk is present, it invokes ROM BASIC.

2. If successful, the master fixed disk boot record is given control and it examines the partition table imbedded within it. If one of the entries indicates a "bootable" (active) partition, its boot record is read (from the partition's first sector) and given control.

3. If none of the partitions is bootable, ROM BASIC is invoked.

4. If any of the boot indicators is invalid, or if more than one indicator is marked as bootable, the message **Invalid partition table** is displayed and the system enters an enabled loop. You may then insert a system diskette in drive A and use system reset to restart from diskette.

5. If the partition's boot record can't be successfully read within 5 retries due to read errors, the message **Error loading operating system** appears and the system enters an enabled loop.

6. If the partition's boot record does not contain a valid "signature" (see "Boot Record/Partition Table"), the message **Missing operating system** appears, and the system enters an enabled loop.

Note: When changing the size or location of any partition, you must ensure that all existing data on the disk has been backed up (the partitioning process will "lose track" of the previous partition boundaries).

Boot Record/Partition Table

A fixed disk boot record must be written on the first sector of all fixed disks, and contains:

1. Code to load (and give control to) the boot record for 1 of 4 possible operating systems.

2. A partition table at the end of the boot record. Each table entry is 16 bytes long, and contains the starting and ending cylinder, sector and head for each of 4 possible partitions, as well as the number of sectors preceding the partition and the number of sectors occupied by the partition. The "boot indicator" byte is used by the boot record to determine if one of the partitions contains a loadable operating system. FDISK initialization utilities mark a user-selected partition as "bootable" by placing a value of hex 80 in the corresponding partition's boot indicator (setting all other partitions' indicators to zero at the same time.) The presence of the hex 80 tells the standard boot routine to load the sector whose location is contained in the following 3 bytes. That sector will be the actual boot record for the selected operating system, and it will be responsible for the remainder of the system's loading process (as it is from diskette). All boot records are loaded at absolute address 0:7C00.

The partition table (with its offsets into the boot record) is as follows:

Offs	Purpose	Head		Sector	Cylinder
1BE	Partition 1 begin	boot ind	H	S	CYL
1C2	Partition 1 end	syst ind	H	S	CYL
1C6	Partition 1 rel sect	Low word		High word	
1CA	Partition 1 # sects	Low word		High word	
1CE	Partition 2 begin	boot ind	H	S	CYL
1D2	Partition 2 end	syst ind	H	S	CYL
1D6	Partition 2 rel sect	Low word		High word	
1DA	Partition 2 # sects	Low word		High word	
1DE	Partition 3 begin	boot ind	H	S	CYL
1E2	Partition 3 end	syst ind	H	S	CYL
1E6	Partition 3 rel sect	Low word		High word	
1EA	Partition 3 # sects	Low word		High word	
1EE	Partition 4 begin	boot ind	H	S	CYL
1F2	Partition 4 end	syst ind	H	S	CYL
1F6	Partition 4 rel sect	Low word		High word	
1FA	Partition 4 # sects	Low word		High word	
1FE	Signature	hex 55	hex AA		

Technical Information

When shipped by IBM, the 10-megabyte fixed disks are formatted with 512-byte sectors at an interleave factor of 6 (17 sectors per track, 4 heads per cylinder.) They contain no data or boot records.

The boot indicator byte must contain 0 for a non-bootable partition, or hex 80 for a bootable partition. Only one partition can be marked bootable.

The "syst ind" field contains an indicator of the operating system that "owns" the partition. Each operating system can "own" only one partition.

The system indicators are:

hex 00 - unknown (unspecified)
hex 01 - DOS

The 1-byte fields labelled "CYL" contain the low-order 8 bits of the cylinder number—the high order 2 bits are in the high order 2 bits of the "S" (sector) field. This corresponds with ROM BIOS Interrupt hex 13 (disk I/O) requirements, to allow for a 10-bit cylinder number.

The fields are ordered in such a manner that only 2 MOV instructions are required to properly set up the DX and CX registers for a ROM BIOS call to load the appropriate boot record (fixed disk booting is only possible from the first fixed disk on a system, whose BIOS drive number (hex 80) corresponds to the boot indicator byte.)

All partitions are allocated in cylinder multiples and begin on sector 1, head 0. EXCEPTION: the partition which is allocated at the beginning of the disk starts at sector 2, to account for the disk's master boot record.

The number of sectors preceding each partition on the disk is kept in the 4-byte field labelled "rel sect." This value is obtained by counting the sectors beginning with cylinder 0, sector 1, head 0 of the disk, and incrementing the sector, head and then track values up to the beginning of the partition. Thus, if the disk has 17 sectors per track and 4 heads, and the second partition begins at cylinder 1, sector 1, head 0, the partition's starting relative sector is 68 (decimal)—there were 17 sectors on each of 4 heads on 1 track allocated ahead of it. The field is stored with the least significant word first.

The number of sectors allocated to the partition is kept in the"# of sects" field. This is a 4-byte field stored least significant word first.

The last 2 bytes of the boot record (hex 55AA) are used as a signature to identify a valid boot record. Both this record and the partition boot records are required to contain the signature at offset hex 1FE.

The Master disk boot record will invoke ROM BASIC if no indicator byte reflects a "bootable" system.

When a partition's boot record is given control, it is passed its partition table entry address in the DS:SI registers.

System programmers designing a utility to initialize/manage a fixed disk must provide the following functions at a minimum:

1. Write the master disk boot record/partition table to the disk's first sector to initialize it.
2. Perform partitioning of the disk—that is, create or update partition table information (all fields for the partition) when the user wishes to create a partition. This may be limited to creating a partition for only one type of operating system, but must allow repartitioning the entire disk, or adding a partition without interfering with existing partitions (user's choice).
3. Provide a means for marking a user-specified partition as bootable, and resetting the bootable indicator bytes for all other partitions at the same time.

Appendix H.
EXE File Structure and Loading

The .EXE files produced by the Linker program consist of two parts:

- Control and relocation information
- The load module itself

The control and relocation information, which is described below, is at the beginning of the file in an area known as the *header*. The load module immediately follows the header. The load module begins on a sector boundary and is the memory image of the module constructed by the linker.

The header is formatted as follows:

Hex Offset	Contents
00-01	hex 4D, hex 5A—This is the LINK program's *signature* to mark the file as a valid .EXE file.
02-03	Length of image mod 512 (remainder after dividing the load module image size by 512).
04-05	Size of the file in 512-byte increments (*pages*), including the header.
06-07	Number of relocation table items that follow the formatted portion of the header.
08-09	Size of the header in 16-byte increments (*paragraphs*). This is used to locate the beginning of the load module in the file.

Hex Offset	Contents
0A-0B	Minimum number of 16-byte paragraphs required above the end of the loaded program.
0C-0D	Maximum number of 16-byte paragraphs required above the end of the loaded program.
0E-0F	Offset of stack segment in load module (in segment form).
10-11	Value to be given in the SP register when the module is given control.
12-13	Word checksum—negative sum of all the words in the file, ignoring overflow.
14-15	Value to be given in the IP register when the module is given control.
16-17	Offset of code segment within load module (in segment form).
18-19	Offset of the first relocation item within the file.
1A-1B	Overlay number (0 for resident part of the program).

The relocation table follows the formatted area just described. The relocation table is made up of a variable number of relocation items. The number of items is contained at offset 06-07. The relocation item contains two fields—a 2-byte offset value, followed by a 2-byte segment value. These two fields contain the offset into the load module of a word which requires modification before the module is given control. This process is called *relocation* and is accomplished as follows:

1. A Program Segment Prefix is built following the resident portion of the program that is performing the load operation.

2. The formatted part of the header is read into memory (it's size is at offset 08-09).

3. The load module size is determined by subtracting the header size from the file size. Offsets 04-05 and 08-09 can be used for this calculation. The actual size is downward adjusted based on the contents of offsets 02-03. Note that all files created by pre-release 1.10 LINK programs *always* placed a value of 4 at that location, regardless of actual program size. Therefore, we recommend that this field be ignored if it contains a value of 4. Based on the setting of the high/low loader switch, an appropriate segment is determined at which to load the load module. This segment is called the *start segment*.

4. The load module is read into memory beginning the start segment.

5. The relocation table items are read into a work area (one or more at a time).

6. Each relocation table item segment value is added to the start segment value. This calculated segment, in conjunction with the relocation item offset value, points to a word in the load module to which is added the start segment value. The result is placed back into the word in the load module.

7. Once all relocation items have been processed, the SS and SP registers are set from the values in the header and the start segment value is added to SS. The ES and DS registers are set to the segment address of the Program Segment Prefix. The start segment value is added to the header CS register value. The result, along with the header IP value, is used to give the module control.

Appendix I. Running Compilers and Assemblers

Using Compilers and Assemblers With Fixed Disk

The following is a summary of how to run the IBM Personal Computer compilers and the IBM Personal Computer Macro Assembler with the Disk Operating System (DOS) Version 2.00. For the purposes of this appendix, the word *compile* will also refer to assemble.

1. Make sure to back up your original language diskettes using the techniques described within this book and/or the respective language books.

2. Make sure all source code you will compile is in the current directory of your disk.

3. Most of the language processors may be located in any directory at compile time (see exception list at the end of this appendix).

4. When Compiling you should use the command lines as described in your language book. You can modify the drive specifier in the command line to indicate the location of the particular language processor.

5. When Linking, use the IBM Personal Computer Linker, Version 2.00, provided with DOS Version 2.00. The instructions within this book and your language book will provide the necessary information for linking.

6. When running a program (.BAT, .COM, or a .EXE file) it is not always necessary for that program to be in the current directory (see the PATH command in Chapter 6 of this book). However, if the program requires another file at runtime; such as a data file or a Common Runtime Module, then those files must be in the current directory at runtime.

7. It is possible for COMMAND.COM to be overwritten in memory. Therefore, it is usually helpful to have a copy of COMMAND.COM in the root directory of the drive from which DOS was started.

Exceptions

- All IBM Personal Computer Language Products:
  - All files accessed by your program must be in the current directory at runtime.
  - Any Common Runtime Module used by your program must be in the current directory at runtime.
- IBM Personal Computer BASIC Compiler:
  - BASRUN.EXE, if used at runtime, must be in the current directory.
- IBM Personal Computer COBOL Compiler:
  - COBOL.COM and its overlays must be in the current directory at compile time.
  - To compile, you must use the command line and indicate the drive location of the overlays with the /C parameter.
  - COBRUN.EXE must be in the current directory at runtime.
- IBM Personal Computer Pascal Compiler:
  - Pascal requires a hex update (see "Pascal Hex Patch" in Appendix J) in order to run this compiler from a drive other than Drive A.
  - PASKEY must be in the current directory at compile time.

Notes:

Appendix J.
Running the Pascal Compiler

Using Pascal Hex Patch With Fixed Disk

The IBM Personal Computer Pascal Version 1.00 Compiler requires that a special file called PASKEY be in the current directory on diskette drive A. The following hex update can be used so that the Pascal compiler will look on the default drive for this PASKEY file.

The following describes how to update your PAS1 diskette. Follow the instructions carefully. If you make a mistake don't panic. Just start again. All the things you enter are in double wide characters.

To update your PAS1 diskette, you need a blank formatted diskette. Put your DOS diskette in diskette drive A and the blank formatted diskette in diskette drive B. You must use the DOS DISKCOPY command to make an exact copy of your PAS1 diskette.

After the DISKCOPY program is started and you see the message **Strike any key when ready**, insert the ORIGINAL copy of the Pascal PAS1 diskette in diskette drive A. It must be the diskette from your IBM Personal Computer Pascal Compiler package or an exact copy of it. Anything else will not work. After you insert the diskettes, press the Spacebar to start the disk copy:

```
A>DISKCOPY A: B:

Insert source diskette in drive A:

Insert target diskette in drive B:

Strike any key when ready


Copy another (Y/N)?N
```

Place your DOS diskette back in diskette drive B. You now have an exact copy of your Pascal PAS1 diskette in diskette drive B. You can now apply the update to the PAS1 diskette with the DOS DEBUG program on the DOS diskette.

The DEBUG program prompt is "-." All the things that you type are after the "-."

When you are responding to the "-" prompt, you must press the Enter key. Note that the *xxxx* in the data displayed line will be filled with the appropriate memory addresses:

```
A>DEBUG
-LDS: 100 1 A5 1
-DDS: 177 LA
xxxx:0177  41-3A 50 41 53 4B 45 59 20      A:PASKEY
xxxx:0180 20
```

If you do not see the above line of data after the "DDS:177 LA" command then you did something wrong and you should start again. You can start again by typing Q followed by Enter. This returns you to DOS:

```
-EDS: 177 "PASKEY  "
-WDS: 100 1 A5 1
-Q
```

(Remember to enter the two blanks after PASKEY.)

You are now back in DOS, you should now recreate your working PAS1 diskette from the updated diskette and mark both diskettes so you know which are the updated ones. The PAS1.EXE file on the updated PAS1 diskette will now look for the PASKEY file in the current directory on the default drive.

Notes:

Appendix K. Considerations for Using Applications

If you have any of the following applications, please refer to the appropriate section in this appendix for additional information about using these applications with DOS 2.00:

- Arithmetic Games 1 and 2
- Asynchronous Communication Support Version 2.00
- EasyWriter Versions 1.10 and 1.20
- Fact Track
- PFS: File
- PFS: Report

- SNA 3270 Emulation and RJE Support Version 1.00
- Typing Tutor
- VisiCalc Version 1.10
- 3101 Emulator Version 1.00

Arithmetic Games 1 and 2

When using the arithmetic games with DOS 2.00, you must have a mimimum of 128K bytes of memory. In addition, if you have a single-drive IBM Personal Computer, you should use the single-drive setup procedure.

Asynchronous Communications Support Version 2.00

To use this application with DOS 2.00 you must have a minimum of 128K bytes of memory.

To install this application in a subdirectory on a fixed disk, use the following procedure. This procedure assumes the following:

- The name of the subdirectory is ASYNC.
- The fixed disk is the default drive.
- The fixed disk drive is drive C.
- The root directory of the fixed drive contains DOS 2.00.

The Procedure

1. Insert the Asynchronous Communications Support diskette into drive A.

2. Enter the following commands in order to create the subdirectory ASYNC and to copy the files from drive A to drive C:

   MD \ASYNC
   Create the subdirectory

   CD \ ASYNC
   Operate from it

   COPY A:*.* C:\ASYNC
   Copy all files on the application diskette

   COPY C:\BASIC.COM C:\ASYNC
   Copy BASIC.COM to the subdirectory

   ERASE UPDATE*.BAT
   Not needed

   ERASE MESSAGE
   Not needed

3. You may now run the Asynchronous Communications Support program by doing the following:

   CD \ASYNC
   Operate from the subdirectory

   AUTOEXEC
   Run the program

   CD \
   Return to the root directory

4. You may run the file conversion program, FILECONV, by doing the following:

   CD \ASYNC
   Operate from the subdirectory

   FILECONV
   Run the FILECONV program

   CD \
   Return to the root directory

Notes:

1. The use of the subdirectory name of ASYNC is an example only. You may use any name you wish.

2. All file transfers will be to and from the files in the subdirectory unless the printer is the destination.

3. This program tests the length of a filespec, and does not accept subdirectory names as part of a filespec.

4. You may rename AUTOEXEC.BAT if you choose.

EasyWriter

To use EasyWriter 1.10 and 1.20 with DOS 2.00 it is recommended your IBM Personal Computer have a minimum of 128K bytes of memory.

To copy DOS 2.00 to your EasyWriter program diskette, use the following procedure:

1. Follow the instructions in the "Startup" section of your EasyWriter manual for copying DOS to your EasyWriter program diskette. While copying DOS 2.00 to the program diskette, you may see the message:

   INSUFFICIENT DISK SPACE (or)

   SECTOR NOT FOUND ERROR WRITING DRIVE B
   ABORT, RETRY, IGNORE?

   If you see the second message above, enter I (for ignore).

   Note: For a system with a fixed disk, follow the instructions for copying DOS 2.00 to your EasyWriter program diskette for a one-drive system.

   Due to insufficient diskette space, the DOS 2.00 FORMAT.COM utility was not transferred to your EasyWriter diskette. To use the FORMAT.COM utility, insert the DOS 2.00 diskette in drive A and a blank diskette in drive B. Then type: FORMAT B: and press the enter key. This will format the diskette in drive B.

3. EasyWriter recognizes drive A and drive B as the drives to read or write from for EasyWriter files. Because the fixed drive is initially defined as drive C, you must reassign its name to be drive B. Enter ASSIGN C=B.

4. You can have your fixed disk automatically assigned to be drive B everytime you load your EasyWriter program. Use the following procedure to do this:

   a. Copy DOS 2.00 onto your EasyWriter program diskette following the instructions given above.

   b. Start DOS 2.00 from the fixed disk. The DOS prompt C> should appear.

   c. Put your EasyWriter program diskette into drive A.

   d. Enter COPY ASSIGN.COM A:

   e. Enter COPY CON A:AUTOEXEC.BAT

   f. Enter DATE

   g. Enter TIME

   h. Enter ASSIGN B=C

   i. Enter EW

   j. Press the F6 key. Now press the Enter key.

   The fixed disk can now be used to store and read your EasyWriter data files. Every time you load your EasyWriter program the fixed disk will automatically be assigned as drive B.

Fact Track

To use Fact Track with DOS 2.00 you need 128K bytes of memory.

If you have DOS 1.10, you should use it to follow one of the setup procedures documented in the Fact Track user manual. The setup procedure copies the necessary DOS programs onto the Fact Track program diskette. From then on, start the Fact Track program diskette in drive A.

If you have only DOS 2.00, and if you have an IBM Personal Computer with one or two disk drives, you should start up DOS in drive A. Enter the date and time as requested. When the system responds:

```
A>
```

you enter:

```
BASICA
```

When the system responds:

```
OK
```

remove the DOS diskette from drive A and insert the Fact Track diskette into drive A and close the door.

Enter:

```
RUN "COLOR
```

Do this every time you want to run the Fact Track program.

If you have DOS 2.00, but do not have DOS 1.10, and an IBM Personal Computer with a fixed disk that contains DOS or BASICA, you should start up DOS from the fixed disk. Enter the date and time as requested. When the system responds:

```
C>
```

Put the Fact Track program diskette in drive A and close the door.

You enter:

```
A:
```

When the system responds:

```
A>
```

You enter:

```
C:BASICA COLOR
```

Do this every time you want to run the Fact Track program.

PFS:File

To use PFS:File with DOS 2.00 you need a minimum of 128K bytes of memory.

Follow the instructions in Part 1 of the Introduction "Getting Ready to Use File." Use your DOS 2.00 diskette whenever the instructions ask for the DOS diskette.

Using PFS:File with the IBM Fixed Disk

You can use PFS:File with the IBM fixed disk in two different ways. First, you can store your files on the fixed disk, thus allowing larger files to be stored, and faster access to forms in the stored files. Second, you can copy the PFS:File program to the fixed disk and then load it from there. This allows you to load the program faster, without using the program diskette.

Storing a PFS:File on the Fixed Disk

You can store a PFS:File on the fixed disk provided you include the drive identifier for the fixed disk, and any needed subdirectory name, as part of the file name. If you specify the fixed disk drive as the default drive, then it is not necessary to include the drive identifier as part of the file name.

Copying PFS:File to the Fixed Disk

Follow these steps to copy the PFS:File program to the fixed disk:

1. Insert the DOS 2.00 diskette into drive A and turn on your IBM Personal Computer. Enter the date and time when the computer asks you to do so.

2. When the DOS prompt appears, remove the DOS 2.00 diskette and replace it with the PFS:File program diskette.

3. Follow the instructions in Appendix D of the PFS:File manual called "Setting Up a Serial Printer and the Work Drive" to run the setup program. Change the work drive name to the drive name of your fixed disk. For example, if your fixed drive is drive C, then the work drive item on the setup menu should be drive C.

4. If you have a serial printer, enter the correct values for the other items on the setup menu.

5. Press the F10 key to complete the setup program. Then enter FTRANS in response to the DOS prompt.

The in use lights will come on alternately as the program is copied from the diskette to the fixed disk. The copy is placed in the root level directory. The DOS prompt reappears when the copy is complete.

If you have more than one fixed disk, the copy will be made to the drive whose name is last in the alphabet. For example, if two drives are named C and D, the copy will automatically be made to drive D.

Error Conditions

The following error conditions might occur during the copy procedure.

Message	Explanation	Corrective Action
CAN'T COPY PROGRAM FILE	Your program diskette has been damaged.	Try the copy procedure with the backup copy of the program diskette.
	Your fixed disk is improperly formatted. (also applies to the CREATE message)	Copy to diskette any files on the fixed disk. Then reformat the fixed disk, and try the copy procedure again.
CAN'T CREATE PROGRAM FILE	Your root level directory is full.	If you have unnecessary files in the root level directory, delete them and start the copy procedure again.
NAME ERROR	Possible DOS error.	Reload DOS 2.00 and start the copy procedure again.
WRONG VERSION	Cannot find the fixed disk.	Make sure that your fixed disk is properly connected, and that you are using DOS 2.00.

Running the PFS:File Program from a Fixed Disk

To run the PFS:File program from a fixed disk, make sure that you have followed the instructions under "Copying PFS:File to the Fixed Disk." Then in response to the DOS prompt, type and enter the drive identifier for the fixed disk followed by the name of the program. For example, to run the File program from drive C, enter C:FILE. If the default drive is C, you need only enter FILE in response to the DOS prompt.

Changing Settings When Using the Fixed Disk

To change the work drive or the information stored for your serial printer after copying the PFS:File program to the fixed disk, you need to insert the PFS:File program diskette into drive A and run the setup program from that diskette. Use the COPY command to copy the file named IBMSETUP.PFS from the diskette in drive A to the fixed disk.

PFS:Report

To use PFS:Report with DOS 2.00 you must have a minimum of 128K bytes of memory, and you should do the following:

1. Copy the sample file called STAFF, which is on the Report program diskette, to a blank formatted diskette or to a fixed disk. See "Formatting Diskettes" and "Copying a File" in Appendix C of the PFS:Report manual.
2. Erase the STAFF file from the Report program diskette using the ERASE command of DOS 2.00.
3. Now follow the instructions in Part 1 of the Introduction in the PFS:Report manual "Getting Ready to Use Report." Use your DOS 2.00 diskette when the instructions ask for the DOS diskette.

Using PFS:Report with the IBM Fixed Disk

You can use PFS:Report with the IBM fixed disk in two different ways. First, you can store your files on the fixed disk, thus allowing larger files to be stored, and faster access to forms in the stored files. Second, you can copy the PFS:Report program to the fixed disk and then load it from there. This allows you to load the program faster, without using the program diskette.

Storing a PFS:File on the Fixed Disk

You can store a PFS:File on the fixed disk provided you include the drive identifier for the fixed disk, and any needed subdirectory name, as part of the file name. If you specify the fixed disk drive as the default drive, then it is not necessary to include the drive identifier as part of the file name.

Copying PFS:Report to the Fixed Disk

Follow these steps to copy the PFS:Report program to the fixed disk:

1. Insert the DOS 2.00 diskette into drive A and turn on your IBM Personal Computer. Enter the date and time when the computer asks you to do so.

2. When the DOS prompt appears, remove the DOS 2.00 diskette and replace it with the PFS:Report program diskette.

3. Follow the instructions in Appendix D of the PFS:Report manual called "Setting Up a Serial Printer and the Work Drive" to run the setup program. Change the work drive name to the drive name of your fixed disk. For example, if your fixed drive is drive C, then the work drive item on the setup menu should be drive C.

4. If you have a serial printer, enter the correct values for the other items on the setup menu.

5. Press the F10 key to complete the setup program. Then enter RTRANS in response to the DOS prompt.

   The in use lights will come on alternately as the program is copied from the diskette to the fixed disk. The copy is placed in the root level directory. The DOS prompt reappears when the copy is complete.

   If you have more than one fixed disk, the copy will be made to drive whose name is last in the alphabet. For example, if two drives are named C and D, the copy will automatically be made to drive D.

Error Conditions

The following error conditions might occur during the copy procedure.

Message	Explanation	Corrective Action
CAN'T COPY PROGRAM FILE	Your program diskette has been damaged.	Try the copy procedure with the backup copy of the program diskette.
	Your fixed disk is improperly formatted. (also applies to the CREATE message)	Copy to diskette any files on the fixed disk. Then reformat the fixed disk, and try the copy procedure again.
CAN'T CREATE PROGRAM FILE	Your root level directory is full.	If you have unnecessary files in the foot level directory, delete them and start the copy procedure again.
NAME ERROR	Possible DOS error.	Reload DOS 2.00 and start the copy procedure again.
WRONG VERSION	Cannot find the fixed disk.	Make sure that your fixed disk is properly connected, and that you are using DOS 2.00.

Running the PFS:Report Program from a Fixed Disk

To run the PFS:Report program from a fixed disk, make sure that you have followed the instructions under "Copying PFS:Report to the Fixed Disk." Then in response to the DOS prompt, enter the drive identifier for the fixed disk followed by the name of the program. For example, to run the Report program from drive C, enter: C:REPORT. If the default drive is C, you need only type and enter REPORT in response to the DOS prompt.

Changing Settings When Using the Fixed Disk

To change the work drive or the information stored for your serial printer after copying the PFS:Report program to the fixed disk, you need to insert the PFS:Report program diskette into drive A and run the setup program from that diskette. Use the COPY command to copy the file named IBMSETUP.PFS from the diskette in drive A to the fixed disk.

SNA 3270 Emulation and RJE Support Version 1.00

To use SNA 3270 Emulation and RJE Support Version 1.00 with DOS 2.00 you need a minimum of 128K bytes of memory.

To install this application in a subdirectory, SNA, on a fixed disk, use the procedure described below, which assumes the following:

- The fixed disk is drive C.
- The fixed disk drive is the default drive.
- The root directory in the fixed disk contains DOS 2.00.

The Procedure

1. Insert the SNA program diskette into diskette drive A.
2. Edit the files 3270COPY.BAT and SRJECOPY.BAT to change all drive B references to drive C.

3. Enter the following commands to create the subdirectory, SNA, and to copy the needed files from drive A to drive C.

   MD \SNA
   Create the subdirectory

   CD \SNA
   Operate from it

   A:3270COPY
   Copy required files

   A:SRJECOPY
   Copy required files

   COPY C:\BASIC.COM C:\SNA
   Copy a needed file

4. You may now run the SNA program by entering the following commands:

   CD \SNA
   Operate from the SNA subdirectory

   programname
   Enter the correct program name as specified in the SNA manual in response to the DOS prompt

   CD \
   Return to the root directory

   Notes:

   1. The use of the subdirectory name SNA is as an example only. You may use any name you wish.

   2. All file transfers for SRJE are to and from files in the SNA subdirectory.

   3. The SNA program does not accept subdirectory names as part of a filespec.

Typing Tutor

To use Typing Tutor with DOS 2.00 it is recommended that your IBM Personal Computer have at least a minimum of 128K bytes memory. If you have a single diskette drive system, you should use the single drive setup procedure.

VisiCalc Version 1.10 by VisiCorp.

To use VisiCalc Version 1.10 with DOS 2.00 it is recommended that your IBM Personal Computer have a minimum of 128K bytes of memory. The following are procedures to put DOS 2.00 on your VisiCalc program diskette, and to make your program diskette self starting for fixed disk. These procedures assume the following:

- You have at least one diskette drive and one fixed disk.
- You are familiar with loading DOS 2.00 from your fixed disk.
- You are aware that VisiCalc supports a maximum of two secondary storage devices. Only one of these two devices may be accessable during operations.

Putting DOS 2.00 on Your Program Diskette

Follow these steps to put DOS 2.00 on your VisiCalc program diskette and to make that diskette self starting:

1. Start DOS 2.00. You should see the DOS prompt C>.
2. Remove the write protect tab from the VisiCalc program diskette.
3. Place the program diskette into drive A.
4. Enter COPY COMMAND.COM A:
5. Enter SYS A:

6. Enter COPY ASSIGN.COM A:

7. Enter COPY CON A:AUTOEXEC.BAT

8. Enter DATE

9. Enter TIME

10. Enter ASSIGN B=C (or any other drive designator)

11. Type VC80 (space) (press F6) (enter)

12. Remove the program diskette and replace the write protect tab on your program diskette.

The fixed disk can now be used to store and retrieve your VisiCalc data files. Everytime you load your VisiCalc program the fixed disk will be assigned as drive B.

3101 Emulator Version 1.00

To use the 3101 Emulator Version 1.00 with DOS 2.00 requires a minimum of 128K bytes memory.

Use the procedure described below to install the 3101 Emulator in a subdirectory, EM3101, on the fixed disk. The procedure assumes the following:

- The fixed disk is drive C.
- The fixed disk is the default drive.

The Procedure

1. Insert the 3101 Emulator program diskette into drive A.
2. Enter the following commands to create the subdirectory, EM3101, and to copy the needed files from drive A to drive C:

   MD \EM3101
   Create the subdirectory

   CD \EM3101
   Operate from it

   COPY A:*.* C:\EM3101
   Copy all files on the 3101 Emulator program diskette

   ERASE COPYFILS.BAT
   Not needed

   ERASE AUTOEXEC.BAT
   Not needed

   CD \
   Return to root directory

3. You may now run the 3101 Emulator by entering the following commands:

   CD \EM3101
   Operate from the EM3101 subdirectory

   IBM3101
   Run the 3101 Emulator program

   CD \
   Return to the root directory

4. You may run the file conversion program, FILECONV, by entering the following commands:

   CD \EM3101
   Operate from the EM3101 subdirectory

   FILECONV
   Run the file conversion program

   CD \
   Return to the root directory

Note:

1. The use of the subdirectory name EM3101 is as an example only. You may use any name you wish.

2. All file transfers will be to and from files in the EM3101 subdirectory,

3. The 3101 Emulator does not accept subdirectory names as part of a filespec.

Notes:

Appendix L. Keyboard Support

Introduction

This appendix contains important information on the keyboard support files on your DOS diskette. If you have not already done so, you should become familiar with "Loading DOS" earlier in this book before reading this section.

There are two main parts to this appendix. The first part, through "Setting Up a Program Diskette to Use a Keyboard Routine," gives the unfamiliar user an overview of the keyboard support files on the DOS diskette. The user will also be instructed in setting up an application diskette, using one of the files. If you will be typing exclusively with the United States keyboard, you will not need to set up any of your application program diskettes with these keyboard files. You can use them as-is, following only any setup procedures which may have come in the application program manual. You can also use these procedures to change a diskette setup from one language to another. For programmers, the part titled "Programming Considerations" provides technical data on operation and use of the files.

Overview

The keyboard routines are separate files on the DOS diskette. These files allow you to set up your IBM Personal Computer keyboard for any one of six languages. The keyboard routines can be used with your own programs, and with some application programs designed for use with DOS.

The keyboard files can be copied from the DOS diskette to any diskette formatted for DOS files. This makes it possible to put any keyboard configuration on your program diskettes, so the DOS diskette does not have to be the first one loaded every time you start a program. The DOS diskette has a file on it which will transfer the appropriate language file to an application program and cause it to be started everytime the application diskette is started.

Setting Up a Program Diskette to Use a Keyboard Routine

Before you set up an application program diskette, there is some information you need to know. The keyboard files work only with DOS. They can only load automatically if the application program diskette is set up to start automatically. The following steps will help you determine if you can make an application program diskette load your keyboard automatically, then will show you how to do it.

Checking for Automatic Program Loading

First start DOS using your National DOS diskette. If you are not sure how to do this, refer to the "Starting DOS" section in this book.

To determine if the program diskette will start automatically, place the diskette in drive A, and type:

DIR (enter)

This will show you the "diskette directory." It is like a catalog of the files on the diskette. Look at the directory for a file called AUTOEXEC.BAT.

> Note: If you need to stop the screen from scrolling hold the Ctrl key and press the NumLock key, then release both. After examining the screen, press the space bar to continue scrolling.

If the diskette has this file, it passes the first test, so continue.

If the diskette does NOT have the AUTOEXEC.BAT file, it is not set up for automatic loading, and you should use it as-is.

Checking for Available Space on the Program Diskette

The keyboard file takes up some space on the diskette. You need to know if there is enough room on the diskette to add the keyboard file. Before you can determine this, the diskette may need some other files from DOS. These files are usually transferred to the diskette using a procedure in the manual which came with the program. Before you go any further, you should perform any first time procedures in your application program manual, then return to this point in the procedure.

Now that you have all of the files on your program diskette that are required to make it operational, we'll find out if there's enough room for the keyboard file.

This check is different for a system with one and two diskette drives, so follow the procedure which fits your system.

For a Two Drive System:

Make sure your National DOS diskette is in drive A, DOS is running (the A> prompt is displayed) and your application program diskette is in drive B.

Type: **CHKDSK B: (enter)**

Your display will look similar to the example below:

```
179712 bytes total disk space
 19968 bytes in 3 hidden files
124416 bytes in 23 user files
 35328 bytes available on disk

655360 bytes total memory
389376 bytes free
```

The fourth line of the display tells you how much free space is still available on the program diskette (in drive B). If the number is greater than 1800, then you have enough room to put the keyboard file on your diskette and you can continue with this section. Otherwise, skip the rest of this procedure and use the diskette as-is.

For a One Drive System:

Make sure your National DOS diskette is in the drive and DOS is running (the A> prompt is displayed).

Type: **CHKDSK B: (enter)**

DOS will display the following:

```
Insert diskette for drive B: and strike
any key when ready
```

Remove the DOS diskette from the drive, and insert your program diskette. Then press any key.

Your display will look similar to the example below.

```
160256 bytes total disk space
 12800 bytes in 3 hidden files
121856 bytes in 32 user files
 25600 bytes available on disk

131072 bytes total memory
118672 bytes free
```

The fourth line of the display shows total free space left on the diskette. If the number on that line is greater than 1800, there is room for the keyboard file and you can continue with this section. Otherwise, skip the rest of this procedure and use the diskette as-is.

Setting Up the Diskette for Your Keyboard

IMPORTANT: Follow this procedure ONLY if your application diskette has passed all of the above tests.

1. Put the National DOS diskette in drive A and start DOS. Answer the DOS date and time prompts.

2. When the DOS prompt A> is displayed, type:

   ```
   BASIC KBPGM (enter)
   ```

   The following will appear on your display:

   ```
   1=USA 2=Francais 3=Deutsch 4=Italiana 5=Español
   6=English 0=exit ?
   ```

3. Select the number which corresponds to the language for which you wish to set up your diskette.

   Note: If you are setting up a diskette for U.S.A. English, then press the correct key (1) and go to step 4 now.

   The following message will be added to the display:

   Check that the National copy of the DOS diskette is in drive A Press any key to continue

   The DOS diskette is still in the drive, so press any key.

4. The following message will be added to the display:

   Replace DOS diskette with PROGRAM diskette in drive A Press any key to continue

5. Remove your DOS diskette from the drive. If there is a write protect tab covering the write protect notch on your program diskette, remove it, then put the diskette in drive A.

   Press any key.

   The diskette drive light will come on for a few moments, then the screen will display:

   PROGRAM diskette now contains keyboard routine and AUTOEXEC
   A>

   Your program diskette will now start up your selected keyboard automatically every time you put it in the drive and turn the computer on. At this time, replace the write protect tab on your program diskette.

Keyboard Templates

Your IBM Personal Computer Keyboard comes with keybuttons for English. A template located at the back of the DOS manual also has the layout of the United States keyboard printed on it. You will probably not need this template for regular typing on the IBM Personal Computer. It is provided primarily so that you will have the list of additional characters which are printed on the back of the template. In addition, the templates for each of the six keyboard layouts are printed on pages at the rear of the Guide to Operations manual. You may reproduce these pages to create a template for any other keyboard layout you need. Refer to the IBM Personal Computer Guide to Operations manual for a discussion of keyboard usage.

Selecting the Keyboard Format

Keyboard format is selected automatically when you start DOS, after you have performed the National DOS diskette creation exercise in the "Starting DOS" section in this book. Once you have created your National DOS diskette, it (or a copy) should be the one with which you always start DOS. The master DOS diskette that came with the DOS manual is only used to create diskettes, not to start and run DOS. The keyboard selection is made by the National DOS diskette file called AUTOEXEC.BAT. Similarly, if you have set up an application program diskette using the exercise earlier in this appendix, it also has an AUTOEXEC.BAT file. This file makes the keyboard selection before starting the program. These two AUTOEXEC.BAT files are similar, but not identical. For more information on AUTOEXEC.BAT, see "Batch Processing" in the DOS Commands section of this manual.

Programming Considerations

When using DOS, the user creates a National DOS diskette as instructed in "Starting DOS". This procedure builds an AUTOEXEC.BAT file on a copy of the master DOS diskette. This AUTOEXEC.BAT file invokes a keyboard routine, then returns control to DOS. You may wish to configure your own DOS diskette which does not use this function. The list below shows all keyboard and National DOS related files on the master DOS diskette, and gives a brief description of each. You may choose to make a copy of the DOS master diskette which does not contain all of these files, depending on your needs.

KEYBIT.COM	Italian keyboard routine
KEYBGR.COM	German keyboard routine
KEYBFR.COM	French keyboard routine
KEYBSP.COM	Spanish keyboard routine
KEYBUK.COM	English keyboard routine

WTDATIM.COM An assembler program which prompts for date and time in native language. This file opens the AUTOEXEC.BAT file on the diskette in the default drive and checks which keyboard file is invoked by the AUTOEXEC, then uses this information to determine which of six sets of messages to display when prompting for date and time. If there is no keyboard file invoked or there is no AUTOEXEC.BAT file, the default prompt is in English.

KBDOS.BAS A BASIC program — loaded by master DOS AUTOEXEC.BAT to prompt for language and create National AUTOEXEC.BAT.

KBPGM.BAS A BASIC program — sets up an application program diskette to automatically start a keyboard routine when program is loaded.

AUTOEXEC.BAT The DOS master AUTOEXEC.BAT — used to create the National diskette.

When invoked, the routine is loaded into user memory starting at the lowest portion of available user memory. The BIOS interrupt vector which services the keyboard is changed by the routine to redirect the CPU to the section of user memory where the new keyboard routine now resides. Each keyboard routine takes up approximately 1.8KB of read/write memory, and has lookup tables which return ASCII values unique to each language. The scan and ASCII codes are passed to the system or application program in the AH and AL registers, respectively. These are the same registers used by the ROM BIOS keyboard routine. Exiting back to DOS after loading of the rountine leaves the routine in memory (INT 27H — terminate but stay resident).

Once the keyboard interrupt vector is changed by the DOS keyboard routine, the interrupt is always serviced by the routine in read/write memory. Return to the U.S. English keyboard format is available by holding the Ctrl and Alt keys and pressing the F1 key at the same time. This does not change the interrupt vector back to the BIOS location. In this case the interrupt is still processed by the read/write routine, but the lookup to convert scan codes to ASCII codes is done in the ROM locations. Similarly, holding the Ctrl and Alt keys and pressing F2 causes a return to the read/write lookup tables.

Special Characters

The DOS keyboard routines have provisions for special characters which are language and programming oriented.

Programming oriented characters are provided on the front faces of the keybuttons on the French, Italian, Spanish and German keyboards. Access to these characters is through the use of multiple shift states. The keyboard routines for these countries monitor the states of the Alt and Ctrl keybuttons. If both keys are found to be in the shift state when a keyboard interrupt occurs, the scan code for the appropriate front face character is returned in register AL.

Some language oriented characters are "constructed" by the keyboard routine by using the "dead key" characters. When the keyboard submits a scan code which is designated as dead key by the routine, the dead key information is stored in a buffer until another keyboard interrupt occurs. At this time the scan code for the second interrupt is compared to a table of allowed characters for dead key operations. If a match is found, the two scan codes are combined to form the ASCII code for the proper character, which is returned in register AH. If a match is not found (the character to be accented is not an allowed character), the dead key character code is returned alone first, followed by the second character code. In the case of the diaresis (¨), an invalid second character will result in a box (■) being displayed (254 decimal), and a "beep" will sound.

Table of Allowed Dead Key Combinations

Germany	á é Ê í ó ú à è ì ò ù
France	ä Ä ë ï ö Ö ü Ü ÿ â é î ô û
Spain	ä Ä ë ï ö Ö ü Ü ÿ á é É í ó ú à è ì ò ù â ê î ô û
UK	dead key not supported
Italy	dead key not supported.

Special Considerations When Using DOS Keyboard Support

The nature of the DOS function which loads the keyboard routines is such that it is possible to "stack" more than one routine in user memory. If this occurs, user memory is wasted, as only the last routine invoked can be active. When a keyboard routine is invoked by DOS, it is placed at the lowest location of unused read/write memory, and left resident. If a second keyboard routine is loaded at any time without first performing a system reset (Ctrl, Alt, Del), it is placed in memory at the NEXT unoccupied location (which would leave the current routine still resident). This results in wasted user memory space, since the original routine cannot be returned to in memory. The user should always perform a system reset before changing to another DOS keyboard routine in order to avoid this waste of memory.

Character Sets for the Color/Graphics Adapter

The IBM Color/Graphics Adapter supports three different resolution modes. In the lowest resolution mode (text) the entire character set (from ASCII 0 to 255) can be displayed on the screen. In higher resolution modes (i.e., BASIC SCREEN 1 or SCREEN 2), ASCII values 128 through 255 are defined in the 8 by 128 byte table pointed to by interrupt vector 1FH, which is set at power on to a value which does not normally contain the proper data for correct display of these characters. Since some of the characters for non-United States keyboards are ASCII codes above 127, any use of a keyboard routine to display text in these graphics modes will result in some unreadable characters being displayed.

The DOS diskette has a COM file (GRAFTABL.COM) which initializes the Color/Graphics Adapter for full character support in the graphics modes. This file defines an 8 by 128 byte table which contains the data for each of the standard characters in the 128 to 255 range, and puts the vector pointer for this table into the interrupt 1FH. Once loaded, the table remains resident until system reset, and takes up approximately 1.3KB of user memory.

To invoke this routine, the command GRAFTABL is specified after the DOS prompt A>. After the routine is loaded, it responds with the message GRAPHICS CHARACTERS LOADED, then control is returned to DOS.

Since some user memory is taken up by this routine, it should only be used when text support for the high-end ASCII characters is required in the graphics mode.